University of Canterbury

Master's Thesis

Real-Time Hybrid Tracking for Outdoor Augmented Reality

Author:
Samuel Williams

Supervisors:
Dr. Richard Green
Dr. Mark Billinghurst

A thesis submitted in fulfilment of the requirements for the degree of Master of Science in Computer Science

at the

Human Interface Technology Laboratory New Zealand
Department of Computer Science and Software Engineering

March 2014

Declaration of Authorship

I, Samuel Williams, declare that this thesis titled, 'Real-Time Hybrid Tracking for Outdoor Augmented Reality' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: 22 October 2013

UNIVERSITY OF CANTERBURY

Abstract

Human Interface Technology Laboratory New Zealand
Department of Computer Science and Software Engineering

Master of Science in Computer Science

Real-Time Hybrid Tracking for Outdoor Augmented Reality

by Samuel WILLIAMS

Outdoor tracking and registration are important enabling technologies for mobile augmented reality. Sensor fusion and image processing can be used to improve global tracking and registration for low-cost mobile devices with limited computational power and sensor accuracy. Prior research has confirmed the benefits of this approach with high-end hardware, however the methods previously used are not ideal for current consumer mobile devices. We discuss the development of a hybrid tracking and registration algorithm that combines multiple sensors and image processing to improve on existing work in both performance and accuracy. As part of this, we developed the Transform Flow toolkit, which is one of the first open source systems for developing and quantifiably evaluating mobile AR tracking algorithms. We used this system to compare our proposed hybrid tracking algorithm with a purely sensor based approach, and to perform a user study to analyse the effects of improved precision on real world tracking tasks. Our results show that our implementation is an improvement over a purely sensor fusion based approach; accuracy is improved up to 25× in some cases with only 2-4ms additional processing per frame, in comparison with other algorithms which can take over 300ms.

Acknowledgements

I would like to thank my supervisors Dr. Mark Billinghurst and Dr. Richard Green for their support during this research. Additionally, I'd like to thank Dr. Tadao Takaoka, Ken Beckman, and Leigh Beattie for their friendship and guidance.

Richard, at the start of my research, gave me a clear foundation for my work by suggesting that I focus on the algorithm itself rather than it's implementation on a mobile phone. This was a driving force behind the entire Transform Flow framework which abstracts away the underlying hardware and allows algorithms to be developed and tested reliably on a variety of different platforms.

Mark has an incredible knowledge of augmented reality and has provided a holistic element to my research. While I was just focusing on my algorithm implementation, he recommended that I get a practical demo running on a real phone. This allowed me to demonstrate the viability of my approach with real hardware and perform a user study.

Tad has a mind for algorithms and has always given his time generously when I have questions or need guidance. The depth of his knowledge allowed me to refine my ideas, and brought precision to both my implementation and thesis writeup. He is a true role model for me.

Ken Beckman and Leigh Beattie, and the HIT Lab NZ in general, have provided invaluable support and assistance during my research. Ken has always been supportive of me and my work, while Leigh has always had an open door to discuss ideas and problems. I deeply appreciate the open and friendly environment they have created at the HIT Lab NZ.

Published Work

Samuel Williams, Dr. Richard Green, Dr. Mark Billinghurst. Transform Flow: A Mobile Augmented Reality Visualisation and Evaluation Toolkit, *Image and Vision Computing New Zealand, 2013.* Based on Chapters 2, 5, 6, and 7.

Samuel Williams, Dr. Richard Green, Dr. Mark Billinghurst. Hybrid Tracking using Gravity Aligned Edges, *Computer-Human Interaction New Zealand, 2013.* Based on Chapters 2, 3, 4, 6, and 7.

Contents

List of Figures

List of Tables

I dedicate this work to my wife Coco and my unborn baby.

We seek to improve the world
for those who come ahead of us.

Since the first draft and the final version of this thesis, she has been born.
Welcome, Hinoki Maple Williams, may you one day change the world.

Chapter 1

Introduction

Augmented reality (AR) is a technology that allows virtual content (such as text, pictures, 3D models and sounds) to be blended with images of the real world[5]. AR systems have been an exciting research area for over 40 years. Recently the technology has been deployed on mobile devices[6, 7] and used in outdoor applications[8, 9].

FIGURE 1.1: An example of virtual furniture being overlaid on a video stream from a commercial project, FurniView[1].

Typical outdoor AR systems such as Layar[10], Wikitude[11] rely on the global positioning system (GPS), magnetometer (compass), gyroscope and accelerometer sensors to provide position and orientation information. However, in practice these sensors often have a large degree of error and can be easily affected by local environmental phenomenon[12].

This project investigates how to improve both global and local tracking using sensor fusion and computer vision techniques, within the limitations of current consumer mobile devices.

1.1 Problem

AR systems deal with two fundamental technical challenges: tracking the camera's position and orientation in the real world, and registering virtual object geometry with images taken from the camera. A typical AR system, as shown in Figure 1.2, combines various forms of data from one or more physical hardware sensors along with virtual content to produce a visual experience. The usability and quality of an AR system is directly related to how well the system addresses these challenges.

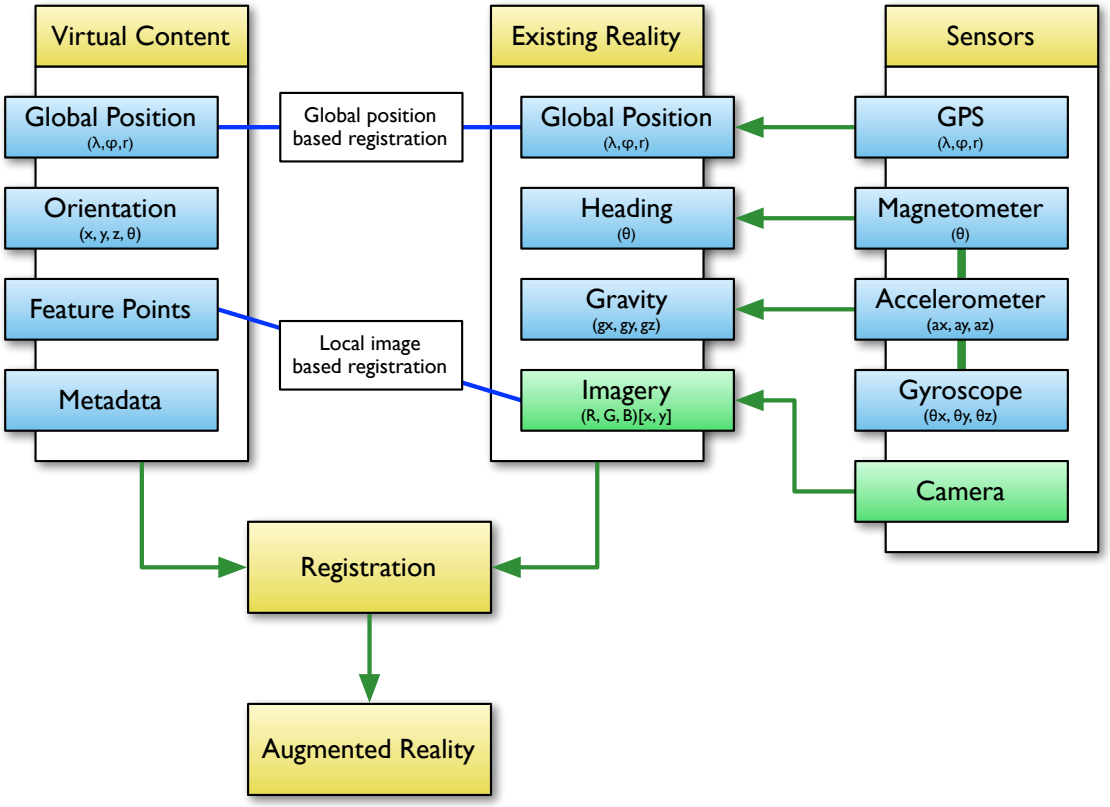

FIGURE 1.2: The conceptual organisation of a typical AR system.

Outdoor AR encompasses the problem of visualising geographically registered data sets with a mobile phone or tablet that includes a camera, a set of inertial sensors (henceforth referred to as sensor data) and some kind of video output. The most challenging part of outdoor AR is tracking the camera pose with respect to a geographical frame of reference[13]. Modern mobile devices provide us with a wide range of inputs, yet, in practice, consumer level sensors have insufficient accuracy for precise tracking (an example of compass accuracy can be seen in figure 1.3).

Tracking accuracy directly affects the quality of outdoor AR applications[14–16]. In particular, knowing the users current position and orientation is critical for aligning

FIGURE 1.3: An example of sensor error in the magnetic compass: a variety of different devices are all measuring a different magnetic north. Ideally, they'd all be pointing in the same direction.

virtual content correctly. A combination of physical sensors and visual data can be used to compute accurate sub-pixel alignment, however the organisation and structure of the data processing will affect the performance and accuracy of the position and orientation calculations.

A significant amount of research has already gone into solving many of these problems, including how to deal with sensor drift using visual information[12] and how to improve orientation estimation using natural feature tracking[17]. However, the majority of this research has been done using custom hardware and with specific applications in mind, and is thus unsuitable for deployment on consumer-level mobile devices.

Our research focuses on adapting existing approaches and designing new algorithms to improve outdoor AR on modern mobile devices. We explore how much the stability and responsiveness of outdoor augmented reality tracking can be improved by combining multiple sensors, with a particular focus on the sensors available in modern consumer mobile phones and tablets. We discuss the process of developing an efficient visual tracking algorithm using motion estimates; this method will leverage sensor readings to minimise the costs associated with feature point detection and alignment, and take inspiration from existing hybrid tracking algorithms where possible.

1.2 Major Contributions

This thesis research is driven by the following key outcomes:

- A sensor fusion based tracking algorithm based on existing methods. This will be used as a baseline implementation for testing and comparative evaluation.
- A novel hybrid tracking algorithm utilising sensor fusion based tracking and computer vision techniques. Developing a fast and efficient algorithm that improves on existing methods is the primary goal of this research.
- A framework for evaluating the accuracy of tracking with structured data sets. Testing the algorithm systematically will allow us to evaluate its performance and accuracy thoroughly.
- An implementation of the algorithm in a software library for the iPhone platform. A practical implementation on mobile hardware will allow us to explore real-world performance and issues.
- A demonstration mobile outdoor AR application based on the software library. This will allow us to evaluate user performance and the qualitative attributes of our implementation.
- Publications describing the implementation and evaluation of the hybrid tracking algorithm.

In Chapter 2 we discuss existing approaches to tracking and registration. Our proposed algorithm is described in detail in Chapter 3 and Chapter 4. Chapter 5 explains the tools we developed to support this research, which were then used to evaluate our proposed algorithm in Chapter 6.

Chapter 2

Background

In 1968, Ivan Sutherland developed the first optical see-through system, which when mounted to a user's head, could respond to changes in position and orientation[18]. It combined both a mechanical arm and ultrasonic transmitters/receivers to calculate the head pose (position and orientation) with low latency. Despite the limitations in computing power at the time, simple wireframe models could be rendered and presented using stereographic projection with separate CRT displays for each eye.

"The WearComp 1", developed in 1980 by Steve Mann[19], is generally recognised as the first wearable computer to include visual augmentation. It featured a camera mounted on the user's head and an eye-level video see-through display. It was primarily used for photography and light painting, and was the first in a long line of prototypes.

Ten years later, the term "augmented reality" is believed to have been coined by Tom Caudell while working at Boeing as a researcher, with the first demonstratively useful system being developed by Louis Rosenberg in 1991 at the U.S. Air Force Research Laboratory[20].

Since then, AR technologies have continued to evolve - and, unsurprisingly, the mobile phone has received an increasing amount of interest from both researchers and commercial ventures as a platform for AR applications[21]. Modern mobile phones are sufficiently powerful enough for a variety of different tracking and registration tasks[22], but despite significant progress, many technical challenges remain. We

investigate the current state of the art, and identify the issues which need to be addressed to produce robust mobile outdoor AR systems.

2.1 Mobile Hardware

Mobile phones present a unique opportunity for AR applications due to their convenient size, configuration and availability (over 400 million smart phones sold in the first half of 2013[23]). A typical modern mobile phone includes one or two cameras, a bright touch screen designed for outdoor use, and a reasonable level of processing power. However, despite significant advances, mobile phones still require specific algorithms and software engineering knowledge to ensure reliable and well-behaved applications (e.g. battery drain, heat output)[21].

Modern AR applications often depend on algorithms which employ image processing and complex interactive 3D graphics overlaying video input. Existing desktop AR applications might not perform well when ported directly to a mobile phone[24, 25], because of the reliance on desktop class hardware and assumptions about connectivity. Conversely, mobile AR applications designed specifically for mobile hardware including the available sensor data and camera/screen configuration would not work on desktop class hardware.

In addition, mobile hardware can be difficult to work with due to the wide range of available devices. Physical size, shape and form factor, camera intrinsics and positioning, sensor accuracies, screen resolutions, processing power, and rendering capability can vary significantly across consumer-level hardware and may present problems when implementing applications that should work seamlessly for as many users as possible.

2.1.1 Adaptive Tracking

Many algorithms naturally have an accuracy/efficiency trade-off. Sometimes, the performance characteristics can be statically adjusted at compile-time, e.g. the input video frame resolution, the search window size, the number of feature points per frame, etc. In addition, some algorithms can be dynamically adjusted at run-time, adapting to the user's behaviour and the level of accuracy required, as shown in

Figure 2.1. By reducing processing requirements, battery usage and heat output can be reduced.

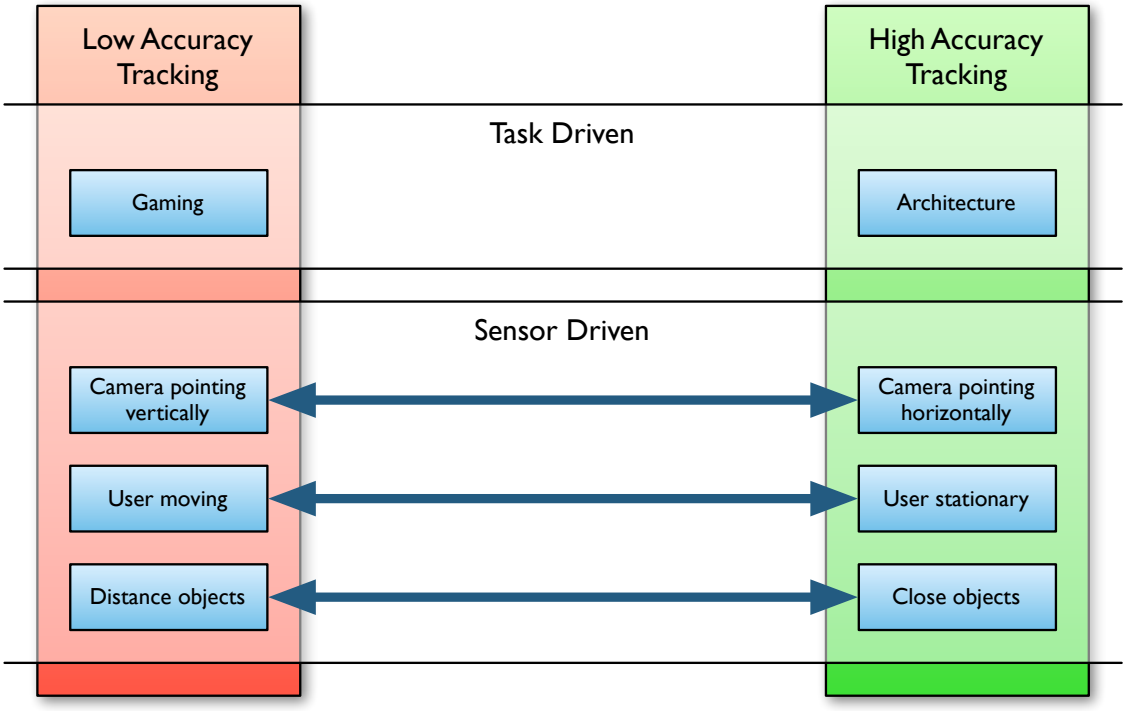

FIGURE 2.1: An overview of the adaptive continuum whereby some situations require more accuracy than others.

Vision processing is a relatively expensive operation when compared to sensor fusion, and if accurate tracking is not required, sensor fusion alone may be sufficient for calculating the camera pose. Urban environments, such as cities, where sensor data including from the GPS and magnetometer may be highly inaccurate[26, 27] could benefit significantly from image based tracking. By observing the error in sensor measurements, image processing algorithms could be used only when required.

2.2 Authentic Experiences

Users have authentic AR experiences when the visual cohesion between the real world and virtual content is maintained precisely and realistically. Tracking and registration errors, including drop outs, misalignment, failed initialisation, and drift, have such an effect on the experience that they make AR systems unusable or unsuitable for practical deployment[13, 16]. Thus there has been a large amount of research with focus on tracking and registration in the last two decades with ongoing work still required to improve accuracy and efficiency of these systems.

Ideally, applications should provide reliable and unrestricted end-user experiences, but this usually requires task-specific algorithms that give the best possible results for a specific interaction, and often fail or work poorly in other situations. Algorithms based on fiducial markers[2], like the one shown in figure 2.2, can accurately align virtual content with real world markers, but the marker must always be clearly visible through the camera, which limits the way the user can interact with the content. Discontinuities like this break the illusion that the augmented content exists in the real world.

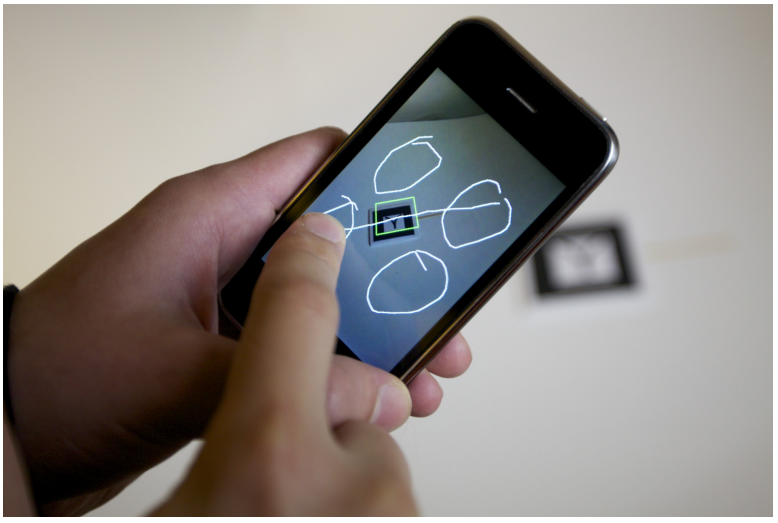

FIGURE 2.2: ARToolKit[2] based fiducial marker tracking using the Dream framework[3]. Although the user can draw anywhere, it can only be rendered correctly when the marker is visible.

Applications which require geographical alignment[28] often focus on computing a global frame of reference (i.e. latitude, longitude). This allows users to experience content registered with real world positions. However, typical consumer level hardware is unable to compute a global frame of reference with sufficient accuracy for precise visual alignment[26, 29]. High quality hybrid approaches[22] are therefore critical for applications that want to provide genuine end-user interactions that depend on both accurate local and global registration.

2.2.1 Practical Usability

It has been identified that tracking and registration are critical for usable AR applications[30]. Unless the objects in the real world and virtual world are aligned with respect to each other, the illusion that the two worlds coexist may be compromised[31]. The human eye can differentiate between a dark and light bar grating

where each bar subtends about one arc minute[32]; similarly, with the availability of high resolution cameras and displays in consumer level mobile devices, the tolerance for visual alignment errors in video see-through displays is continually decreasing.

A recent study has shown that when an application presents multiple interfaces, e.g. a map, a compass and an AR view, users tend to avoid AR visualisations, and visual jitter has been identified as one possible reason for this[16]. Accurate tracking itself is an enabler for different kinds of interactions, e.g. pedestrian navigation, landmark identification, but the quality and performance of the algorithms employed will have a significant effect on the overall usability of an application.

2.3 Visual Tracking and Registration

Purely image based approaches to tracking and registration can be used for computing a local frame of reference over multiple individual frames. Existing research in this area is typically developed for desktop class hardware or specialised robotic platforms[33] making it unsuitable for use on mobile hardware. However, computer vision techniques provide opportunities for improved tracking and registration for outdoor augmented reality and thus we investigate how they can be adapted to improve efficiency.

Optical flow[34–36] is the visual motion of specific features in sequential video frames caused by movement of visible objects or the camera. Optical flow measurements can be used to calculate changes in camera pose and object positions and thus can be used for both tracking and registration related tasks. A set of visual corners or feature points is typically used as an input to typical optical flow. However, even modern algorithms designed for efficient implementation[37–40] performed poorly in our tests on current mobile hardware (detailed results are shown in Table 3.4). Our observations are supported by a performance comparison on high end hardware[4], we republish their results in Table 2.1. Optical flow itself provides sub-pixel accuracy but practical implementations would need to be improved by several orders of magnitude before being efficient enough for real time operation on current mobile hardware.

OPIRA[41], shown in figure 2.3, uses a pre-determined database of feature points and image rectification to locate planar surfaces in real-time. The image rectification provides perspective invariance, which allows the algorithm to register content

TABLE 2.1: Average Processing Time of Visual Descriptor Algorithms for the Computation of 500 descriptors[4]

SIFT	SURF	BRIEF	ORB	BRISK	FREAK
43.45ms	13.43ms	1.43ms	1.36ms	2.11ms	1.09ms

accurately, even at highly acute angles. However, OPIRA doesn't provide support for camera pose tracking except in relation to these specific natural features, and thus isn't suited for dynamic tracking in an unknown environment.

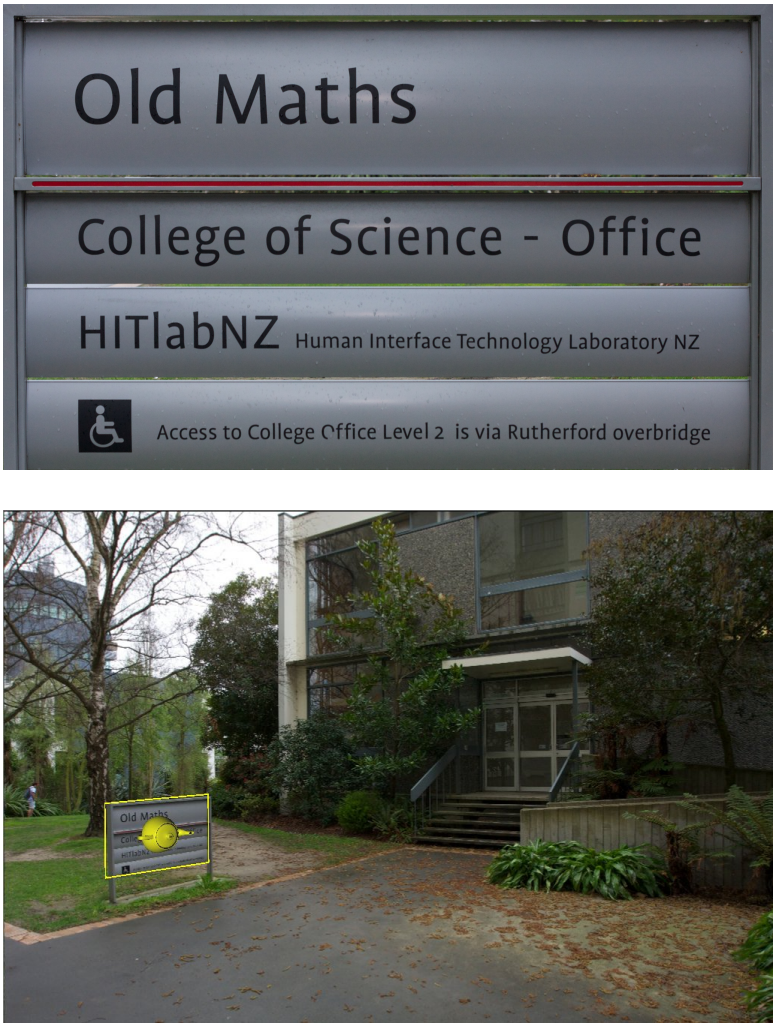

FIGURE 2.3: An example of OPIRA using image based registration to overlay a teapot model on the sign. In this case there is no global frame of reference.

BoWSLAM[42] matches 3-dimensional feature points and is designed to overcome robustness issues with previous wide-area single camera SLAM implementations. It builds a high-level bag-of-words representation of every frame so that new frames can be robustly matched even during very erratic motion, in the presence of moving objects, following large loops, or in other difficult situations where traditional

feature point tracking approaches would fail. However, BoWSLAM was designed for mapping long trajectories. On a small scale, in a static environment, it isn't designed to give the same level of accuracy that you would expect with traditional feature point tracking algorithms.

2.4 Global Tracking and Registration

Global tracking is the process of measuring the camera position and orientation relative to a global frame of reference, such as the Earth-centred Earth-fixed (ECEF) coordinate system. The ECEF frame of reference has it's origin at the centre of the Earth and rotates with the Earth. The X axis passes through the equator at the prime meridian. The Z axis passes through the north pole but it does not exactly coincide with the instantaneous Earth rotational axis. The Y axis can be determined by the right-hand rule to be passing through the equator at 90° longitude[43].

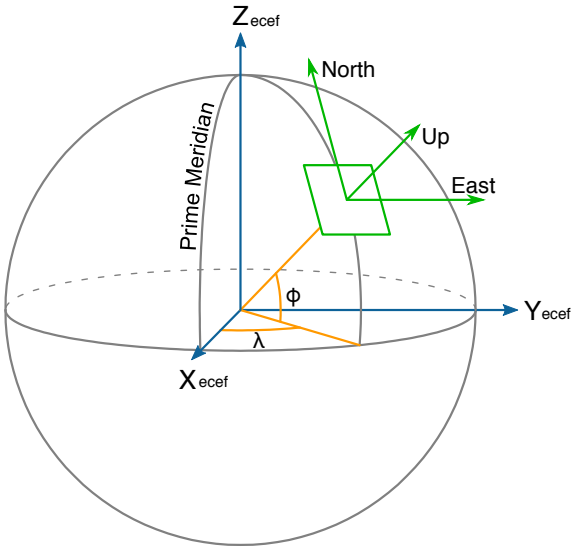

FIGURE 2.4: A global frame of reference, ECEF, can map XYZ to a cartesian East-North-Up (ENU) coordinate system.

The Global Positioning System (GPS) provides latitude, longitude and altitude coordinates using the World Geodetic System standard (established in 1984, thus referred to as WGS84), a grid based geodetic datum that can be used to map coordinates to an ellipsoid body that approximates the surface of the Earth (e.g. ECEF or ENU). Altitude, or more accurately elevation, is used to measure the vertical distance from a reference point, typically the mean sea level (MSL) as defined by WGS84.

Using raw WGS84 coordinates can be complicated for local registration tasks[44]. The WGS84 latitude and longitude values themselves require significant precision for calculations (e.g. double-precision floating point numbers in C) which we observed to be inefficient in certain situations on modern mobile hardware. By converting these into a local frame of reference, e.g. ENU, precision issues can be mitigated and single-precision floating point numbers can be used for positional deltas (i.e. positions relative to the current WGS84 origin).

In addition, physically registered positions on the Earth shift over time with respect to a coordinate system defined by WGS84[45]. This makes WGS84 a poor choice for applications that require precise alignment. Country and region specific geodetic datums (e.g. GDA94 for Australia, NZGD2000[46] for New Zealand) are available which are updated on a regular basis such that coordinates generally match physical features. WGS84 coordinates, including the time at which the measurement was made, can be converted into a local geodetic datum for improved accuracy. However, the complexities of dealing with region specific geodetic datums is also significant, and could still include significant position error.

Gravimetric and magnetic forces can be used to calculate camera orientation in relation to the fixed poles and centre of the earth. The angle between the current direction and a north-south line (i.e. meridian) is known as bearing, and gravity provides a vertical reference line for calculating pitch and roll; these measurements collectively can be referred to as an attitude or pose.

Global registration is the process of positioning content using absolute global coordinates. Given the current global position and orientation in some coordinate system, it is possible to calculate the relative displacement of another global position. Relative positions can be calculated using either spherical coordinates (latitude, longitude, altitude) or cartesian coordinates (ENU) depending on the application.

2.5 Sensors and Computer Vision

Sensor fusion can be used to improve the quality of sensor measurements by intelligently combining sensor data[12]. Mathematical models and filters (such as the complementary filter, Kalman filter or particle filter) can be applied to multiple sensors to improve the measurement of particular physical attributes[47]. However, sensor fusion alone only serves to reduce error towards the minimum possible of a

particular set of sensors, if the sensors include a static bias or dynamic errors to that effect, the output will always be incorrect.

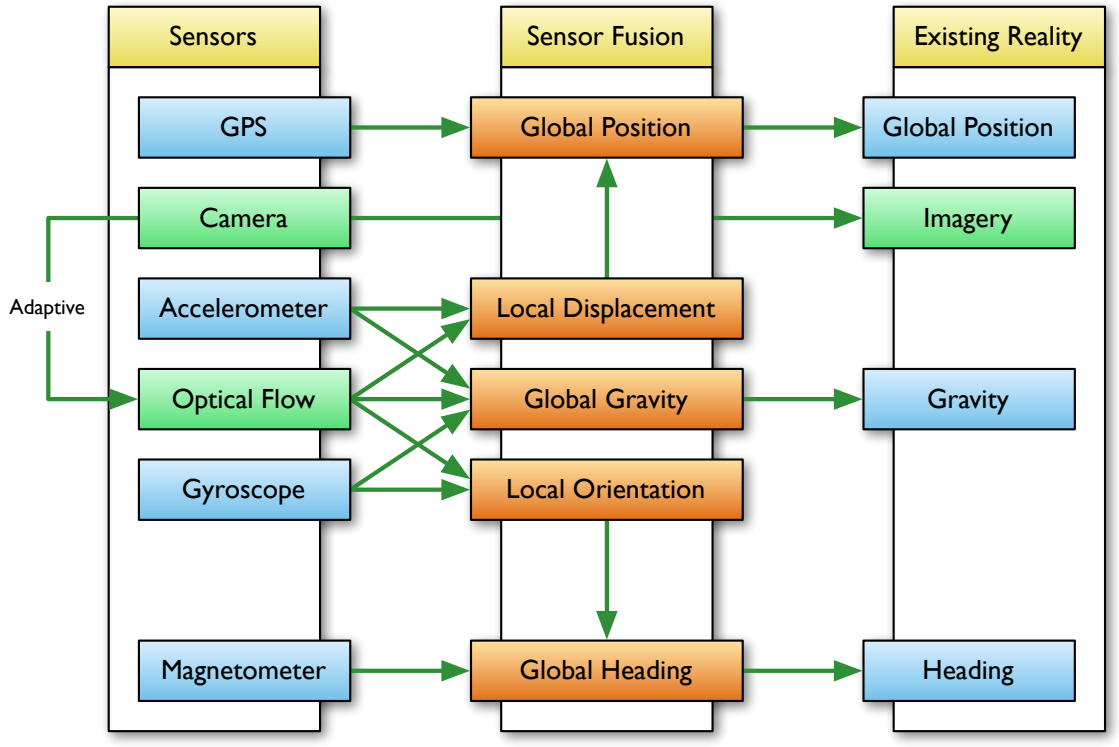

FIGURE 2.5: An overview of how sensor fusion and image processing can be used to improve accuracy.

To improve on this, computer vision techniques have been used to provide robust wide-area tracking on wearable computing and laptop based systems[48, 49], with computations similar to those shown in Figure 2.5. These approaches are typically computationally expensive, requiring dual-core processors or better[50], and large amounts of memory[51], which makes such approaches unsuitable for current mobile devices.

Using vision analysis to reduce errors in inertial sensor measurements is a proven technique[52]. Sensor data is relatively efficient to compute and use, but suffers from a variety of issues[12]. Relative inertial sensors, such as the gyroscope, may drift over time or provide incorrect measurements. Absolute global sensors such as the magnetometer and GPS may provide incorrect measurements due to local interference and usually have a high latency on consumer grade hardware[29]. Image based methods are generally robust in such circumstances, but can be computationally expensive and suffer in cases of visual occlusion, motion blur and rolling shutter artefacts[53]. These types of visual distortions are especially problematic on

mobile devices due to the quality of the cameras typically included. If motion can be accurately measured, it is possible to mitigate both motion blur[54] and rolling shutter artefacts[24]. By exploiting the complementary nature of these different inputs, it is possible to produce a more robust and reliable output[55].

2.5.1 iPhone 4 Sensor Accuracy

The iPhone 4 hardware platform has a variety of sensors, each with absolute and relative error. We measured the typical range for sensor error and latency using practical observations and present our results in Table 2.2, and our measurements are supported by others[26, 27]. Because these sensors are used to calculate camera position and orientation, understanding the nature of the error that exists can allow us to improve the accuracy of any algorithm that depends on these inputs.

TABLE 2.2: iPhone 4 Sensor Accuracy

Sensor	**Absolute Error**	**Relative Error**	**Latency**
GPS (Horizontal)	±10m	±10m	$\leq$ 10s
GPS (Vertical)	±20m	±20m	$\leq$ 10s
Compass	±20°	±5°	$\leq$ 2s
Accelerometer (Angle)	±5°	±1°	$\leq$ 20ms
Accelerometer (Force)	±0.1g	±0.1g	$\leq$ 20ms
Gyroscope	0°	±0.1°/s	$\leq$ 20ms

Absolute error is a measure of how much fixed offset is in all results given by a sensor, such that all measurements within a similar timeframe and position may have the same offset. Absolute error typically exists because of manufacturing defects or local environmental phenomenon, and can often be mitigated by calibrating the specific hardware sensors; in some cases (e.g. the magnetometer) calibration may need to happen frequently, others (such as the accelerometer) may only need to be calibrated once.

Relative error is a measure of how much a sensor may deviate from the actual physical change within a short timeframe. Some sensors (such as the gyroscope) report changes in physical disposition and thus are subject to increasing integration error over time (drift). Relative error can often be corrected by checking one sensor against another to verify changes; if one sensor detects a change, but other sensors disagree, this typically indicates some kind of relative error.

Sensor latency is the time it takes for a physical change to be reported in sensor measurements. Initially, there is some time required for a sensor to respond to a change, and additional time for a sensor to settle on a final value, as shown in Figure 2.6. Sensor latency is primarily a result of hardware sampling and filtering performance. Some sensors can be configured at different sample rates such that physical changes are measured more rapidly; the rate may also affect the amount of sensor drift accumulated over time. When sensors are combined together, the differences in latency may need to be carefully considered.

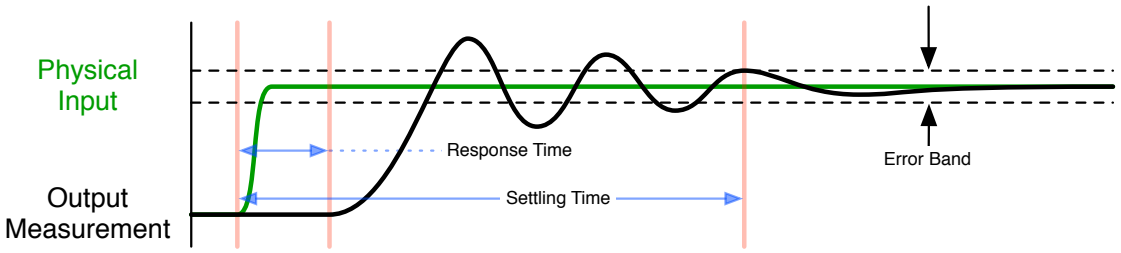

FIGURE 2.6: An example of a step response (hysteresis) where the output requires a certain amount of time to respond and settle to a given input event.

2.6 Local-Global Correspondence Problem

Successful outdoor augmented reality depends on accurate tracking and content registration in relation to a global frame of reference. It is common in mobile outdoor AR applications to use WGS84 coordinates for registering content, and ECEF/ENU coordinates for computing relative positions. In addition, image processing techniques may be used to improve the precision and functionality of both tracking and registration.

However, the terms *position* and *location* should not be used interchangeably[56]. A position specifies a tangible point in a specific coordinate system (e.g. WGS84, ECEF), while a location is an "opportunity to associate meaning" and is identified using a semantic taxonomy based on a very human appreciation of the world.

This confusion affects AR systems in a number of ways: image based tracking algorithms can recognise the same object in different positions, e.g. a magazine cover, which depending on the type of tracking required, could either be a feature or an error. Alternatively, WGS84 coordinates may be used to refer to a precise position, but due to the changing environment (e.g. earthquakes), the coordinate is no longer representing the same physical location, e.g. a door.

Tracking and registration algorithms that seek to be as accurate as possible should consolidate all the available position and orientation metrics in a way that relates to the functionality required. To do this, for a particular tracking and registration task, we must define a specific frame of reference and map each input into that system. The challenge of defining an appropriate mapping can be significant. We refer to the problem of defining and computing this mapping between a local frame of reference and a global frame of reference as the local-global correspondence problem, and define some ways in which it affects outdoor AR systems.

2.6.1 Static Positioning

The most effective sensor for global position estimation on a mobile device is the GPS. Local features can also be identified visually and used to estimate position via a camera, but this requires some pre-established model of the local environment. Calibration errors and physical interference may affect the GPS and cause significant errors. In an urban environment, buildings and walls reflect the GPS signal and cause multi-path signal errors, while overhead foliage can attenuate and scatter the signal. These errors may cause significant problems when correlating visual and sensor based positions.

A region or country specific geodetic datum allows content to be physically registered in a frame of reference with a well defined mapping to a global frame of reference (e.g WGS84 from the GPS). This allows for physically registered locations to be accurately positioned even in the event that the local geography changes over time. However, the geodetic datum must be available and up to date for a specific area, and if not, errors in registration could get worse over time.

Global orientation estimation typically requires input from the magnetometer to compute bearing and fusion between gyroscope and accelerometer to compute gravity. Large scale local magnetic phenomenon (e.g. the composition of the tectonic plates) introduce a static bias into the bearing calculations which means that we have two effective bearings: a magnetic north, and a true geodetic north. Magnetic north points along the magnetic axis of the Earth, while true north points towards the geographic north pole. The difference between these two measurements is referred to as magnetic declination, and varies with both time and place. Therefore, to calculate true north, the position of the device is measured using GPS and a global datum (updated on a regular basis) is used to compute a static bias which

used to adjust the magnetic north vector. Lack of, or error in position measurements, or in the dynamic datum, may affect the quality of the true north bearing significantly.

2.6.2 Pose Tracking

Changes in position can be measured using the GPS, but the update rate on mobile devices is typically slow (e.g. $1Hz+$ updates) and the actual measurements are usually inaccurate, especially in consumer level hardware[29]. The accelerometer can be integrated twice to measure relative changes in position rapidly, but we found that due to noise the error accumulates rapidly which makes it unusable in practice, without compensation from another input. Unfortunately, the latency of mobile phone GPS is not good enough to provide this correction without introducing significant positional errors.

Relative changes in orientation can be accurately measured using the gyroscope. The gyroscope on modern mobile phones is accurate and can be integrated once to get absolute change in orientation with acceptable errors and drift. The magnetometer provides an XYZ magnetic field vector, which can also be used to measure orientation. Combining these two sensors together could reduce gyroscope drift, but the magnetometer is very sensitive and can be easily affected by local magnetic objects which makes it difficult to use in practice.

2.6.3 Content Registration

Accurately registering content against the current global frame of reference can be done precisely by calculating the relative position from the current camera pose. The error in registration will be exactly that of the current global frame of reference. To improve on this, information may be aligned to visual features if they are found within the image data. However, visual features may not be in the same frame of reference, which means that they'd need to be tracked separately. By tracking visually aligned content separately, it can be registered accurately despite errors in the global frame of reference. However, there could be a significant error as content shifts from one frame of reference to another.

Virtual content which includes both a geographical location and visual features (e.g. 2D photographs, 3D pointclouds), can typically be registered more precisely with visual tracking algorithms. However, if visual features are occluded or visual tracking is otherwise impaired, some other visualisation may be more appropriate (e.g. a general marker rather than a detailed overlay). Failures in the visual tracking might create artefacts in the registration of the content, so this would need to be handled carefully.

2.7 Simultaneous Localisation and Mapping

Recently, single-camera Simultaneous Localisation and Mapping (SLAM) algorithms have been investigated for use on mobile devices[57]. Camera based SLAM algorithms typically perform two tasks sequentially: for each video frame, compute the local camera pose from the visual information using a database of feature points (if available), and then expand and refine that database based on new and updated visual features that have been identified.

Panoramic Mapping[22] is a dense SLAM style algorithm that can track changes in camera orientation. It projects visual features on to a cylindrical map and uses this to track camera orientation by matching feature points from the camera input. Sensor data is used to estimate the initial pose of the camera and project the first frame into the cylinder, but is not used in any way to optimise the SLAM process later on. The proposed implementation processed frames at a resolution of 320×240 at 30Hz and uses a dense database of pixels at a resolution of 2048x512. We estimate that a 14 megapixel (56 megabytes of raw RGBA) map would be required to process video at 720p using the same method. The published results[22] indicated a failure rate of 15% with 30 test data sets, primarily due to poor feature point detection.

Parallel Tracking and Mapping (PTAM) is a method of estimating local camera pose in an unknown scene with a single camera, and was first implemented using desktop class hardware[58]. It was later modified to run on a mobile phone[24] with reasonable success. The algorithm splits the processing into two distinct parts that are processed in parallel on different CPU cores. Localisation is performed every video frame (where possible), and the published results[24] indicate that it took approximately 30ms which is barely sufficient for 30Hz operation; and due to limitations in the camera the maximum speed achieved was 15Hz. The mapping

process, running on a background thread, receives sequential frames and builds a database of feature points. Bundle adjustment is used to refine individual feature points and the published research indicates that it took 750ms to process 35 frame bundles, or 20ms per frame. Recent research has been critical of the accuracy of PTAM[59] which reflects our own experiences with the provided implementation. Furthermore, despite running on a mobile phone with inertial sensors, the algorithm was not modified to leverage these in any way.

2.8 Model-based Tracking

Model-based tracking can improve the accuracy of global position by identifying the pose relative to some known structure in the local environment[49]. Models can include a wide variety of features, e.g. buildings, roads, signs, or any other easily recognisable visual feature. By extracting visual features and matching these with the model, the camera pose can be calculated. This process may fail if there are significant changes in the environment: new permanent structures or temporary features may cause significant problems when matching features. Thus, over time, models need to be maintained and kept up to date, making this approach unsuitable for general purpose outdoor augmented reality in unknown environments.

The initialisation problem, where the camera pose is established relative to a given model for the first time, can be computationally expensive. Existing approaches which combine sensors and visual information have been shown to improve the performance and quality of tracking, but in difficult situations took up to 12 seconds to compute the camera pose, which makes them unsuitable for practical mobile use[60].

Recently, dense point clouds[61] have been considered for mobile AR localisation. Panoramic images can be generated from point clouds and used for localisation within the point cloud model; by combining existing image based approaches with the available sensor data on modern mobile phones, the processing costs associated with the initialisation problem can be reduced significantly[62], however, the demonstrated approach still requires 22ms per frame, a significant overhead.

2.9 Gravity Aligned Features

The gravity vector provides a locally consistent frame of reference in which visual information can be processed. Gravity aligned feature descriptors have been shown to improve the reliability and performance of existing feature matching algorithms[63, 64]. However, while the gravity vector improves quality of feature matching, it was not leveraged to improve actual performance of the feature extraction process which is one of the most costly parts of the tracking algorithm.

A recent implementation building on this research confirmed the benefits of gravity aligned features[62], but in practice found that they still suffered from many of the issues affecting non-gravity aligned features, such as difficulty dealing with the types of structural similarity common in urban environments (e.g. buildings with lots of similar windows).

The gravity vector can be used to extract gravity rectified planar surfaces, which can be used to improve tracking and registration of content against flat (horizontal) pages or magazines[65]. We explored whether this would be useful for outdoor AR but concluded that this approach wasn't viable in general. In many cases, the ground plane was not sufficiently visible for accurate tracking and non-planar features could cause significant artefacts in the final unwrapped image.

Modern mobile platforms (including iOS and Android) provide at least a basic level of integrated sensor fusion which includes the gravity vector as an output. Tracking systems which depend on custom hardware or platforms usually implement custom low level sensor fusion algorithms[49]. By depending on the platform's native low level sensor fusion implementation, it is possible to reduce the amount of computation required; dedicated hardware which offloads low-level sensor processing is already available in consumer level hardware and we expect this trend to continue.

2.10 Testing Methodologies

Testing and evaluating tracking algorithms designed for mobile devices is currently unsystematic[24, 57, 66]. This is a big problem for new research which seeks to improve on existing approaches, because it makes it hard to compare results objectively.

Even as recently as 2009, researchers were developing custom devices for outdoor AR[67] research. Tracking algorithms are often structured around poorly specified platforms and obscure hardware[49, 60, 67]; missing details or obsolete hardware make identical reconstruction, and thus comparisons based on published results, impossible.

Data sets and testing tools are often not publicly published[17] which makes it difficult to check whether a new approach is a significant improvement over existing methods. In particular, algorithms that fail on specific edge cases[57] warrant further analysis and study; but without the specific data sets and systematic evaluation tools this is not possible.

Similarly, public data sets commonly used for computer vision evaluation don't include inertial sensor measurements[68–70], which makes them inappropriate for modern mobile AR research[65]. Inertial gravity measurements have been synthesised from ground truth camera poses, which allows some types of hybrid tracking algorithms to be benchmarked on existing data sets[64], however other sensors including gyroscope, magnetometer and compass were not considered. Modern hybrid algorithms may depend on a full range of inertial sensor measurements to operate correctly and efficiently, and thus prior work evaluated with these data sets would not be easily comparable.

Modern consumer mobile phones and tablets provide an excellent variety of sensors in low cost, readily available off-the-shelf packages. This is an ideal platform for many types of AR research, because it is representative of the type of environment algorithms would be expected to work in if deployed to a wider audience. However, despite this level of standardisation, we lack modern tools and data sets for testing and evaluating tracking algorithms on these platforms.

2.11 Source Code Availability

Published research in mobile AR often lacks source code[57, 65]. There are many reasons for this: copyright/licensing/intellectual property restrictions, poor quality code (suitable for evaluation but not for actual use), lack of time to properly release code (documentation, ongoing maintenance, compatibility with multiple platforms); but practically speaking, it makes the development and testing of new approaches

difficult. Implementing a tracking algorithm from scratch is a large undertaking and requires a significant investment of time.

In addition, published data sets and evaluations generated using a specific hardware device quickly become obsolete, as the hardware and sensors are constantly improving and changing. To accurately compare algorithms on consumer level hardware, a working implementation is required, otherwise performance and accuracy cannot be accurately compared. Therefore it is necessary to have access to the source code to perform a realistic comparative analysis of any kind.

The pace of rapid innovation in consumer mobile devices drives changes in the supporting mobile operating systems (OS), including the software frameworks and libraries on which our tracking and registration algorithms are built. Major platforms (e.g. Android, iOS) are completely different in their underlying implementations, such that compiling code for multiple platforms can be a huge burden. There is a culture in commercial AR of only providing the compiled static library, but as hardware changes, these libraries may stop working, and fixing these bugs is practically impossible. If the source code is available and of a reasonable quality, many of these issues at least become addressable.

2.11.1 Existing Projects

We found several open source projects which implement inertial sensor based mobile AR tracking algorithms. However, many of the most promising ones seem unmaintained[71–73]. Other libraries that are maintained, are typically either device specific[74] or application specific[75]. None of the available open source libraries provide specific tools for the development and evaluation of mobile outdoor AR tracking algorithms.

2.12 Summary

We have done a thorough review of existing work in the area of outdoor augmented reality. We have looked at the various tracking and registration methods available, and found that the practical implementations are not fast enough for real world use. In addition, few algorithms use available sensor data effectively and thus may

struggle in cases where visual tracking fails. Because of this, globally registered AR is still common and this is a major problem for usability. By combining local and global tracking methods effectively, many of these problems can be overcome, but the extent to which any proposed algorithm makes an improvement is currently difficult to assess, due to the limited availability of source code and systematic testing tools.

Next, we propose a hybrid tracking algorithm which effectively combines both local and global tracking to overcome these issues.

Chapter 3

Fast Vertical Edge Alignment

We have developed a hybrid tracking algorithm, and a high level overview is shown in Figure 3.1. The vertical edge alignment algorithm is composed of the steps in the green boxes, and we walk through these steps in this chapter.

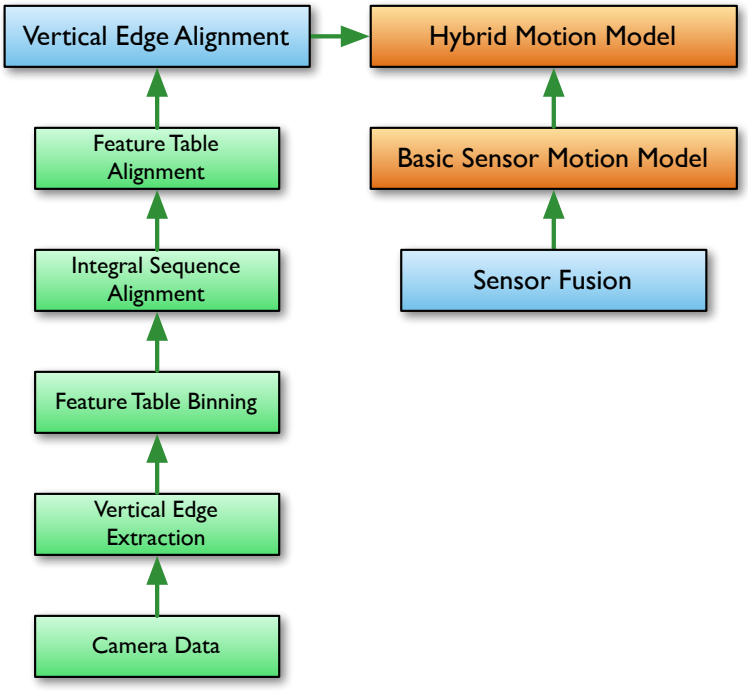

FIGURE 3.1: The structure of our proposed hybrid tracking algorithm.

Our proposed vertical edge alignment algorithm is designed to improve the accuracy of global alignment by combining sensor fusion with image processing to produce a robust bearing measurement. Inter-frame alignment issues are the most noticeable for users of outdoor AR applications[13] and a purely sensor based approach may

suffer from jitter, latency and drift. Our implementation focuses on fixing these issues and has been carefully designed to be both efficient and scalable on current generation mobile hardware.

Our approach to visual alignment is conceptually different from many existing computer vision techniques that essentially depend on the image component as the ground truth[41, 58]. By relying *mostly* on sensor data, we minimise the amount of image processing required for visual alignment. Our approach is validated by our performance results; a high level performance comparison (in Section 3.3.3) of our proposed algorithm with the ORB[38]/Lucas-Kanade Optical Flow[34–36] implementation in OpenCV[76] shows that we are almost 100 times faster with sufficient accuracy.

In this chapter, we discuss our proposed algorithm in detail and evaluate it's performance and accuracy using synthetic tests. The specific mathematical formulation for the alignment computation is discussed in Chapter 4. The tooling used for testing and evaluation is discussed in Chapter 5, with practical real world evaluation presented in Chapter 6.

3.1 Morphological Tracking

Model-based tracking provides many opportunities for improved localisation and registration[49] in comparison to a purely sensor based approach. But creating and updating models for a specific environment takes considerable effort and thus such algorithms are not suitable for general outdoor AR.

By identifying forms and shapes at run-time, pertinent structures in the world can be mapped and used for localisation without prior knowledge of the environment. We refer to this process as morphological tracking, which alleviates the need for a prior model, and instead uses a lexicon of typical environmental shapes (e.g. parallel lines, triangles, rectangles) to optimise tracking and registration tasks.

This is different from pure feature point tracking as we explicitly require that some meaningful structure is quantified and tracked. We believe that this can be implemented more efficiently than feature point tracking in general and provides a more natural way to register content based on real world structures.

For local alignment and registration, some kind of model-based data sets may still be appropriate. Existing approaches are typically data heavy (e.g. point clouds, geometric models with textures), but morphological data sets could be significantly smaller as the data required for the abstract shapes and forms could be reduced and compressed. For mobile applications, sending data over the wireless network is still a significant cost.

3.1.1 Vertical Edges

Our proposed vertical edge alignment algorithm is a type of morphological tracking algorithm. Conceptually, the extracted feature points are an approximate sparse model of the vertical lines in the world. Because of this, we have a high degree of correlation between similar frames, and thus they can used as input to a tracking algorithm. Our implementation uses this statistical approximation to compute the bearing of a frame, given at least one previous frame with a known bearing.

Users of outdoor AR are typically pointing their devices towards the horizon and in such video frames we expect a large number of vertical edges, especially in urban environments where there are buildings and other artificial structures. By tracking vertical lines rather than specific feature points, we can reduce the computational costs involved significantly. Vertical lines can be easily identified using a memory-efficient scanline based search, unlike feature extraction algorithms that must process large amounts of pixel data to extract good edges for tracking purposes.

In addition, vertical edges, parallel to gravity, are the best features to track when measuring translations and rotations perpendicular to gravity. For our proposed algorithm, a good vertical edge is one that has a large luminance gradient perpendicular to gravity and ideally is part of a long continuous edge parallel to gravity. This allows our algorithm to identify the edge easily over several frames.

3.2 Implementation

The vertical edge alignment algorithm computes the precise pixel offset between two video frames, such that vertical edges overlap as precisely as possible. As input, it requires two images, p_1 and p_2, the gravity vector at the time each image was

captured g_1 and g_2, and the estimated translation between the two frames in pixels e, normally computed using the gyroscope. The output sub-pixel offset includes a confidence value which is the number of vertical features aligned correctly. We define the alignment function as follows:

$$\texttt{align}(p_1, g_1, p_2, g_2, e) = \overline{a} \tag{3.1}$$

$$u \leftarrow \texttt{bin}(p_1, g_1) \tag{3.2}$$

$$v \leftarrow \texttt{bin}(p_2, g_2) \tag{3.3}$$

$$o \leftarrow \texttt{FISA}(\texttt{counts}(u), \texttt{counts}(v), e) \tag{3.4}$$

$$i \leftarrow \texttt{max}(0, o) \tag{3.5}$$

$$j \leftarrow \texttt{max}(0, -o) \tag{3.6}$$

$$a_k \leftarrow \sum_k (\overline{v[i+k]}_x - \overline{u[j+k]}_x) \tag{3.7}$$

where $\overline{a}$ means the average distribution of a, `bin` extracts vertical features into gravity aligned histograms, `FISA` (defined in Chapter 4) computes the integral alignment of two sequences, and `counts` returns an array containing the number of vertical features within a particular histogram bin.

The resulting sup-pixel alignment, as computed by Equation 3.7, can be combined with existing sensor fusion based estimates. In our implementation, we used a weighted combination of the sensor fusion input with the image alignment, but we mostly depend on the image alignment (usually more than a 0.95 weighting). In the case that the confidence of the match is low (e.g. less than 3-5 bins matched), we would defer completely to sensor fusion. This worked sufficiently well in our testing, because the image alignment process is typically more robust and as accurate than the fusion of the gyroscope and magnetometer, but in difficult cases, e.g. extreme motion blur, we would defer entirely back compass/gyroscope.

3.2.1 Device Coordinate System

The coordinate system for a phone depends on the physical sensors and the software libraries providing the sensor data. Our algorithm is flexible enough to work with a variety of coordinate systems, but we discuss the algorithm's implementation in terms of what we refer to as the device coordinate system, as shown in figure 3.2.

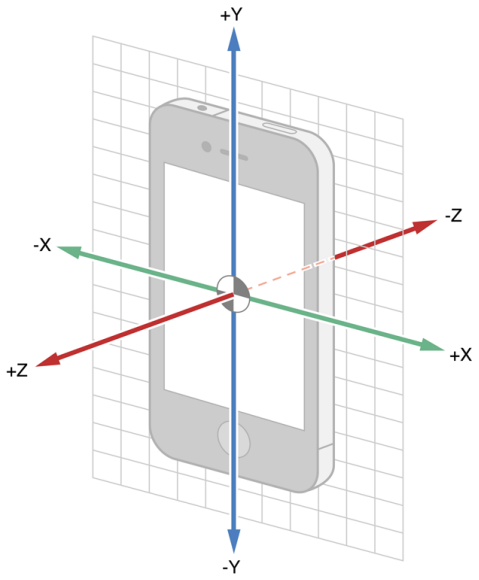

FIGURE 3.2: The device frame of reference for iPhone class hardware.

All sensor data is reported in this frame of reference, but the camera frames are rotated by 90° around the $+Z$ axis, such that in device coordinates, image space $(0, 0) \rightarrow (w, h)$ coordinates would be at $(+Y, +X) \rightarrow (0, 0)$. This is typical for most cameras which are orientated such that the aspect ratio of the video frame matches the aspect ratio of the screen.

3.2.2 Gravity Vector

The gravity vector is sampled at a rate of 60Hz to 120Hz, to ensure that the vector is accurate enough for frames captured at 30Hz. For each frame, the last value for the gravity vector is used to compute the tilt angle, as this is the most accurate and up to date reading.

We have found that the accuracy of the gravity vector with respect to the image plane is excellent, typically on the order of $\pm 0.1°$. In addition, for many typical outdoor AR applications, the user is unlikely to rotate the phone in a way which significantly affects the gravity vector. This makes it an ideal sensor for input to a computer vision algorithm.

The gravity vector can be computed by combining the accelerometer, gyroscope and magnetometer[49]. However, modern mobile platforms include the gravity vector as part of the inertial sensor data. This vector is usually very accurate and the sensor

fusion computation may be accelerated using hardware (e.g. the iPhone M7 motion co-processor), so we use this vector directly rather than calculating it ourselves.

3.2.3 Tilt Calculation

The tilt angle is the rotation of the image frame such that gravity in device coordinates is aligned with the Y axis in camera coordinates (see Figure 3.3). The tilt is only valid for gravity vectors that are not parallel to the camera axis (e.g. not looking directly up or down), and it's computation is dependent on the various coordinate systems of the device's hardware configuration.

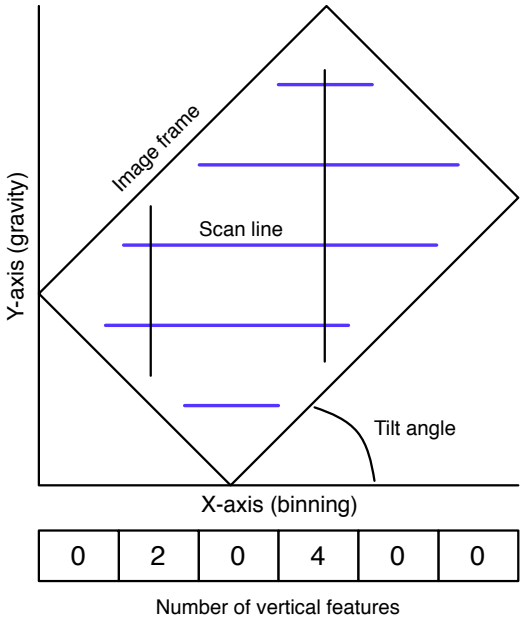

FIGURE 3.3: Vertical edges are extracted and binned relative to gravity.

We define a gravity local coordinate system such that gravity points down $-Z$ and $+X$ points right. This cylindrical mapping is the same as is used in other panoramic tracking algorithms[22] and is useful for most typical outdoor AR applications[1].

3.2.4 Scan Line Extraction

Using the tilt angle, we can compute a set of scanlines perpendicular to the gravity vector (the blue lines in figure 3.3). We compute a rotated bounding box for the

[1]Another option we have considered involves using globally registered scanlines radiating around the user, such that in our gravity local coordinate $Z = 0$ would be the horizon. This may increase the quality of tracking during vertical motion.

image frame and use this to clip a set of horizontal scanlines. We rotate these scanlines back into image space to extract features. The distance between scanlines can be specified and this influences the number of feature points extracted and the amount of pixel data processed.

We use Bresenham's line drawing algorithm[77] to trace these scanlines efficiently (see Figure 3.4) and apply the Laplacian of Gaussian operator sequentially to pixels. The Laplacian operator approximates the 2^{nd} derivative of the pixel intensities along the scanlines and we extract the coordinates of the zero crossings with approximate sub-pixel precision using a fast mid-point calculation.

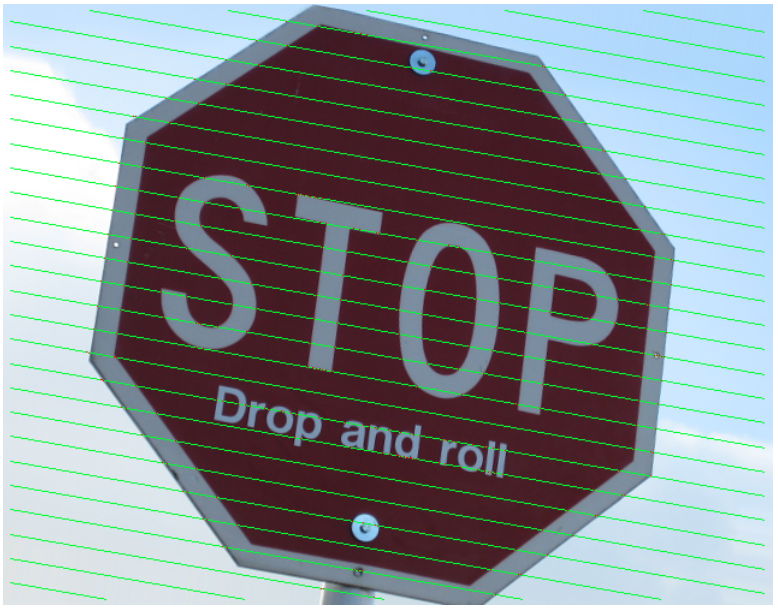

FIGURE 3.4: Scan lines overlaid on an image rotated by 10°.

The Laplacian operator allows us to calculate the mid-points precisely even when edges are not perfectly visible. This allows for accurate key-point detection even when the image includes a large amount of motion blur and/or unfocused regions, which is common for cameras with relatively small sensors and contrast based focusing.

3.2.4.1 Approximate Zero Crossings

Our original algorithm used only adjacent pixels to detect edges - if the difference between the two pixels is above a certain threshold, it would be used for tracking. Despite being very fast, this approach failed to generate many feature points in frames with motion blur.

To alleviate this problem, we implemented an efficient Laplacian of Gaussian (LoG) kernel to detect approximate 2nd derivative zero crossings. We experimented with a number of kernel sizes and variations, but eventually found that $\texttt{LoG}_5$ gave good results both in terms of accuracy and performance:

$$\texttt{LoG}_5(I, x) = \begin{bmatrix} -1 \\ -1 \\ 4 \\ -1 \\ -1 \end{bmatrix} \cdot \begin{bmatrix} I(x-2) \\ I(x-1) \\ I(x) \\ I(x+1) \\ I(x+2) \end{bmatrix} \tag{3.8}$$

where I is an intensity function, e.g. a sequence of pixels, and x is the offset at which we are sampling.

In addition, we improved our feature point extraction to be approximately sub-pixel accurate by calculating the approximate zero crossing. Traditional curve fitting algorithms for finding the precise zero crossing are inefficient[78]. We propose a simple mid-point calculation which does not require additional sampling and computes the sub-pixel zero-crossing with at most an error of ± 0.5 pixels. The efficiency of our approach allows us to process higher resolution data which produces more accurate results with minimal performance impact.

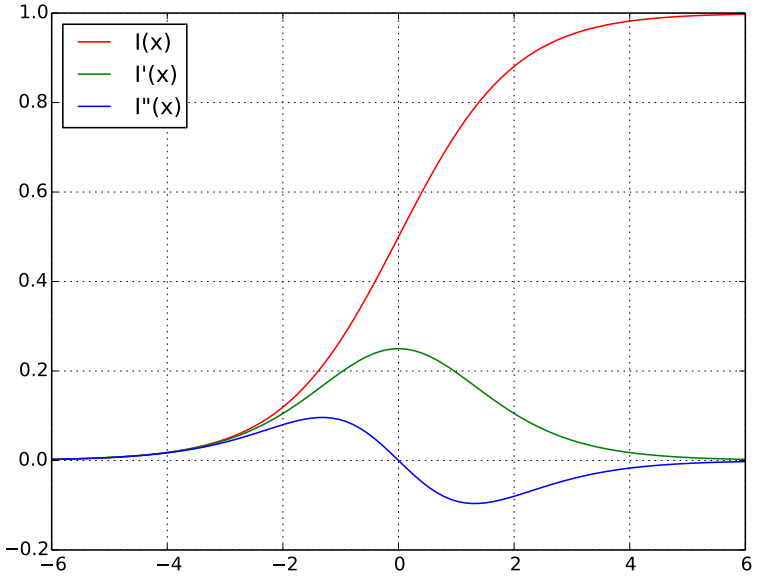

FIGURE 3.5: A typical gradient curve in I, and its 1st and 2nd derivatives.

Given an intensity function for a scanline, $I(x)$, we evaluate the Laplacian of the Gaussian (LoG) kernel at x and $x+1$. This gives us a value for the approximate 2nd

derivative curve at x and $x+1$. If $I''(x)$ is positive and $I''(x+1)$ is negative (or the opposite), there is a zero-crossing point between those two samples[2] (an example is shown in figure 3.5). We can compute the approximate value for x such that $I''(x) = 0$ using a function *midpoint*:

$$\texttt{zerocrossing}(x_1, x_2) = x_1 + (x_2 - x_1) \times \texttt{midpoint}(I''(x_1), I''(x_2)) \tag{3.9}$$

One way to approximate the zero crossing is to take the average value of x_1, x_2:

$$\texttt{midpoint}(a, b) = 0.5 \tag{3.10}$$

However, if the discrete sampling of I'' is not equally balanced around the zero-crossing, which we expect is the most common case, this formulation will produce incorrect results. By modelling the curve as a straight line between the sampled points $\langle x, I''(x)\rangle \rightarrow \langle x+1, I''(x+1)\rangle$, we can compute a more accurate approximation. The following computation calculates the point where this line crosses zero as a relative factor in the range $(0.0 \rightarrow 1.0)$:

$$\texttt{midpoint}(a, b) = \frac{-a}{b-a} \tag{3.11}$$

We used the 2013B data set, shown in Figure 3.7, to evaluate the average midpoint function in equation 3.10 and the approximate midpoint function in equation 3.11. We ran the tests once for each algorithm.

Table 3.1: Midpoint Calculation Accuracy

	Error (°N)			
	Average		Approximate	
Data Set	**S.D.**	**S.E.**	**S.D.**	**S.E.**
2013B0	0.716274	0.102325	**0.647186**	0.0924552
2013B1	1.63702	0.187779	**1.35561**	0.155499
2013B2	1.31868	0.203477	**1.17316**	0.181022

The results in Table 3.1 show a consistent improvement of between $10-20\%$.

[2]We don't consider the case that $I''(x+1) = 0$ because in that situation the mid point can be computed precisely.

FIGURE 3.6: Sample frames from data set 2013A, which features foliage and both near and far vertical edges.

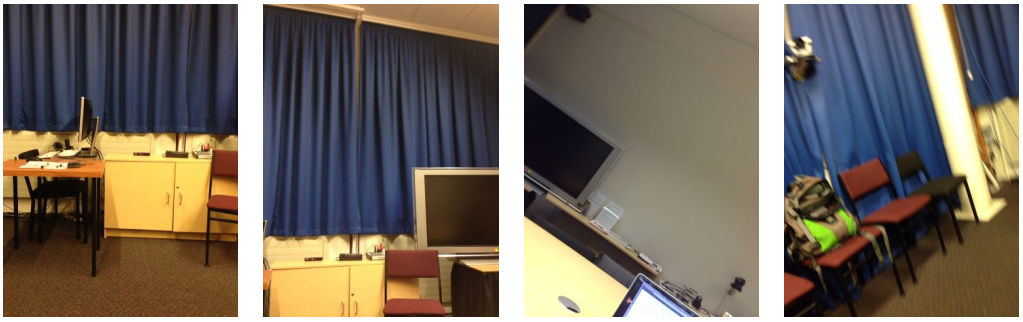

FIGURE 3.7: Sample frames from data set 2013B, which features significant rotations and motion blur.

FIGURE 3.8: Sample frames from data set 2013C, which features several very plain regions.

FIGURE 3.9: Sample frames from data set 2013D, which include moving people and transparent surfaces.

3.2.4.2 Feature Thresholding

Once a zero crossing has been detected, we calculate the local variance and compare it to a threshold. Because the `LoG` function is sensitive to noise, a considerable effort is required to filter good features. We assume that the zero crossing represents a pixel, typically on one side of a gradient, and so we look at the variance between the left side, the centre, and the right side.

$$\texttt{left(I, x)} = \left[\frac{I(x-2)+I(x-1)}{2} - I(x)\right]^2 \tag{3.12}$$

$$\texttt{right(I, x)} = \left[\frac{I(x+2)+I(x+1)}{2} - I(x)\right]^2 \tag{3.13}$$

$$\texttt{variance}(I,x) = left(I,x) + right(I,x) \tag{3.14}$$

This function tries to analyse the local structure of the gradient. It takes a small 2 pixel sample on the left of the zero crossing, and a small 2 pixel sample on the right of the zero crossing. It compares these samples to the value at the zero crossing and sums at the squared difference. For a smooth gradient, we'd expect a similar value for both left and right, but for a step gradient, one side will be big and the other side will be small. This function should compute the same value for the same step size whether or not it is blurred or sharp - in practice it appears to do a good job as seen in Figure 3.10.

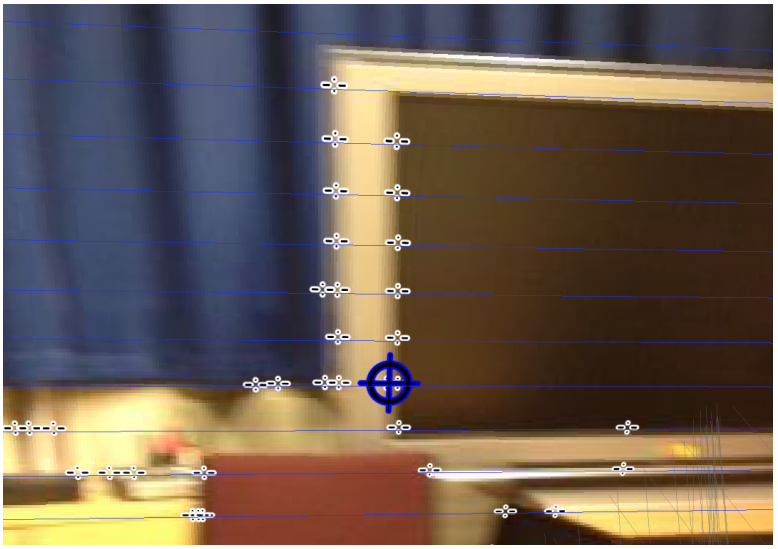

FIGURE 3.10: Features are still detected with good accuracy despite significant motion blur.

We tried a number of different variance functions, and found that the sum of squared difference gave the best accuracy and good performance. It requires no additional sampling of the input image as we use the greyscale values computed for the Laplacian. The amount of noise is tolerable in practice.

3.2.5 Feature Alignment

Our initial feature table implementation tried to extract and match vertical lines efficiently. The green edges in Figure 3.11 show the structure that could be extracted from the feature points. However, after analysing a number of data sets, we decided against this approach. Extracting complete edges robustly across different lighting conditions and motion blur is difficult. Missing feature points could break connected edges incorrectly, and matching up edges in these conditions would require a complex heuristic.

Rather than trying to discretely track connected vertical edges, we implemented a statistical model for alignment. The proposed algorithm considers all identified vertical features, and effectively computes a histogram (an example is given in Figure 3.11) of this data perpendicular to the gravity vector. By aligning the histograms, we compute the pixel alignment along this axis.

3.2.5.1 Feature Table Binning

After extracting the vertical edges, we distribute them into a series of bins (referred to as a feature table), where each bin covers a fixed portion of the X axis in the gravity local coordinate system, as shown in Figure 3.3. Sequential items in a single bin often represent connected vertical lines in the source image.

The size of the bins is flexible; by increasing the bin width, we reduce the total number of bins. This can improve efficiency at the cost of precision. However, if the bin width is too small, visual noise could become problem. In practice, we've identified that a width of 1px or 2px is ideal.

In order to reduce aliasing issues, we ensure that all feature tables have an even number of bins. Different tilt rotations may require a different number of bins, depending on the width of the rotated bounding box. Feature tables with an even number of bins will always align such that the centre of rotation falls precisely

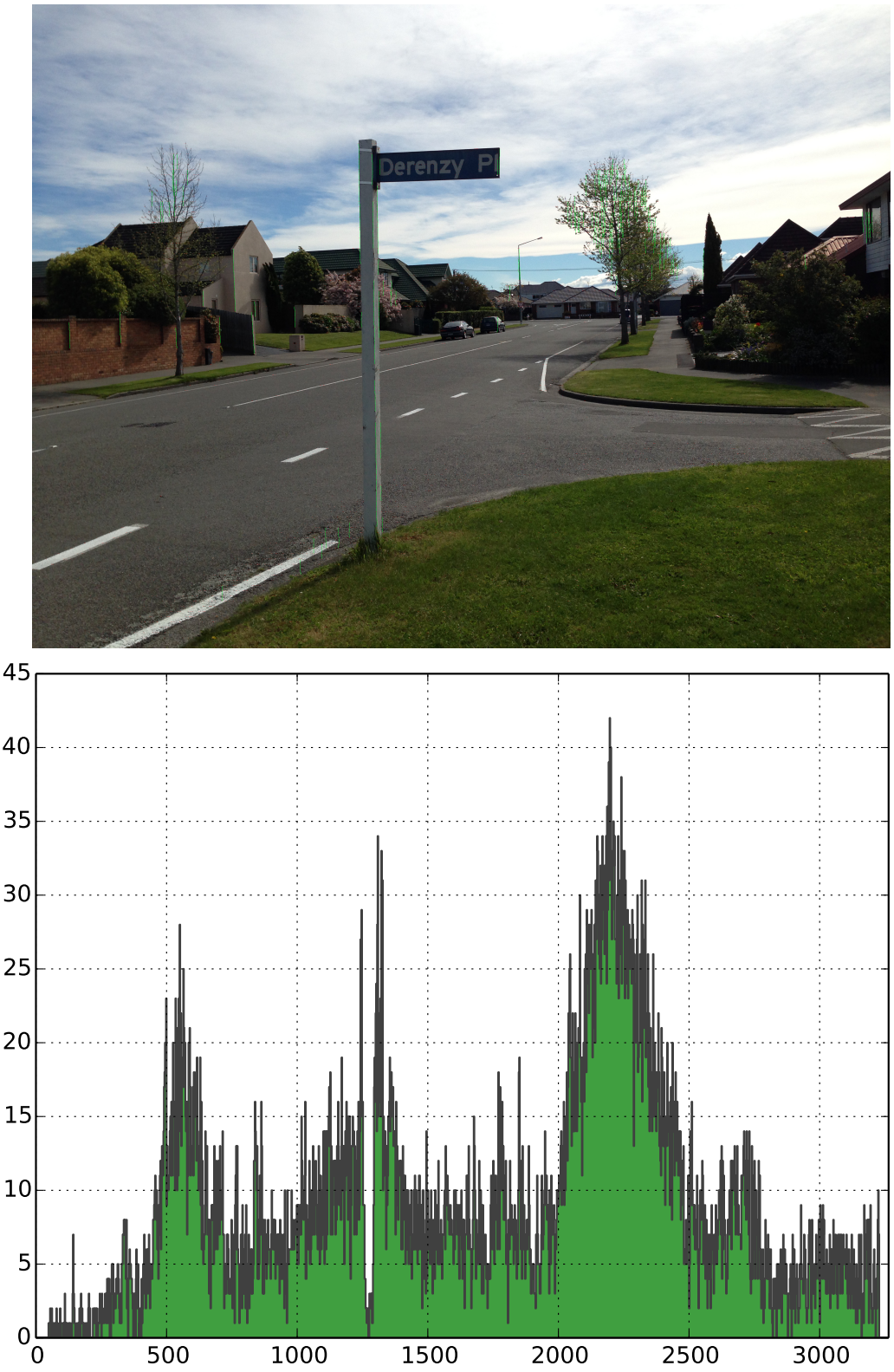

FIGURE 3.11: A histogram of vertical edges in 2px wide bins. The three main peaks from left to right correspond to the left tree, the road sign, and the right row of trees. Approximate vertical lines overlaid in green on the source image. Individual red pixels mark precise feature points.

between two bins, so our proposed algorithm ensures this quality for all feature tables.

3.2.5.2 Table Alignment

We can align two feature tables with bin-level accuracy using the Fast Integral Sequence Alignment algorithm, as discussed in Chapter 4. We use the number of vertical edges in each bin to compute an integral sequence u and v for each feature table. To compute the estimate bin offset, we compute the relative rotation between two frames using the gyroscope and convert this into a bin offset estimate e.

Once we have calculated the bin alignment, $\texttt{FISA}(u, v, e)$, we compute the precise sub-pixel alignment of individual vertical features. For corresponding bins in each feature table, we compute average X position in gravity local coordinates. The difference between the average vertical feature position is computed for all corresponding bins, and used to compute an average displacement in pixels. This displacement is then converted back into a rotation and used as input into the sensor fusion computation.

An alternative approach is to compare the bins in the feature table sequentially. As the feature points are ordered vertically along the Y axis (i.e. gravity), we can compute a precise correspondence between features with similar Y value efficiently. We hoped that this approach would be more tolerant to noise, but instead found that in practice it was not a systematic improvement.

When computing the relationship between two bins, ideally we would like the bins to have a similar and significant number of feature points. This helps to reduce noise. Our proposed algorithm therefore only aligns individual bins that have a fixed minimum number of vertical features.

In practice, this isolates noise from sequential vertical lines, but in difficult tracking situations (e.g. significant motion blur), it might be the best match available, so fine tuning is required. Our confidence of a good match is therefore directly related to the number of bins we could use to compute the alignment. If we have less than a moderate number of corresponding bins, our hybrid tracking algorithm reverts back to a purely sensor based approach.

3.3 Evaluation

We evaluate our algorithm on an iPhone 5 running iOS 7, using the stop sign image from the metaio data set[69], as shown in Figure 3.4. We tested a number of different scenarios to examine the performance of the proposed algorithm and its tolerance to erroneous input.

We also took this opportunity to compare our approach with an existing feature extraction and alignment algorithm to assess the viability of our implementation.

3.3.1 Rotational Alignment

We modified the sample image so that it could be systematically rotated, and generated rotations in 1° increments from −20° to +20° using a script. We used the 0° rotation as the basis and calculated the offset of every other rotation using our proposed algorithm. As the source images only have rotations applied, the translation is expected to be 0 in all cases. As such, the translation estimate supplied to the vertical feature alignment algorithm is set to 0 for these tests.

We varied the distance between scanlines, which directly affects the number of feature points detected, to look at the effect on performance and accuracy. The scanlines are distributed vertically every *dy* pixels. We give the time for feature extraction and alignment processing separately, and the alignment error for all images from −20° to +20°.

Table 3.2: Rotation Performance and Accuracy.

			Output Error (px)		
dy (px)	**Features**	**Alignment**	**Mean**	**S.D.**	**S.E.**
5	12.13ms	303.9µs	-0.0026	0.019	0.003
10	6.29ms	143.3µs	-0.0045	0.028	0.0044
15	4.24ms	109.9µs	-0.0082	0.04	0.0062
20	3.27ms	119.5µs	-0.023	0.046	0.0071
25	2.57ms	125.8µs	-0.0099	0.066	0.01
30	2.08ms	126.5µs	0.16	0.64	0.099

The results in Table 3.2 show that there is a good balance between performance and accuracy when $10 <= dy <= 20$. The number of samples that matched up is dependent on the source image, but we want to ensure a reasonable number of good quality matches. As *dy* is increased, the accuracy is diminished, as expected.

3.3.2 Rotation Noise

Under normal circumstances it would be typical to have a small amount of error in the measured rotation from the gyroscope. We simulate this by adding gaussian noise to the tilt estimate and passing this incorrect data into the alignment algorithm. We fix $dy = 10$ and the bin size is $2px$ for this evaluation, and use the same data set as described in Section 3.3.1.

TABLE 3.3: Rotation with noise.

Input Error		Output Error (px)		
S.D.	**Samples**	**Mean**	**S.D.**	**S.E.**
0.0°	49.2	-0.0049	0.045	0.00071
0.2°	48.3	0.0025	0.044	0.00069
0.4°	46.6	0.013	0.12	0.0019
0.6°	44.8	0.012	0.29	0.0045
0.8°	43.4	-0.032	0.47	0.0073
1.0°	42.3	-0.077	0.58	0.0091

From the results in Table 3.3, we can see that despite significant error in the tilt, the standard deviation is about half a pixel of error. This is fairly reasonable, as the alignment of the feature tables will become progressively worse, proportional to the amount of error in the tilt angle. We also note that the number of matching samples is reduced, indicating a dimished confidence of a good match, which is also expected.

3.3.3 Translation Comparison

We compared our proposed algorithm with an alignment algorithm implemented using optical flow. We were primarily interested in the efficiency of our approach in comparison to existing feature point extraction and correspondence algorithms. We generated a set of test images with fixed offsets from -20px to +20px in 5px increments, and processed them using both implementations.

The results in Table 3.4 show that our proposed algorithm is significantly faster in practice than existing feature extraction and optical flow alignment algorithms. In particular, ORB is considered to be a reasonably fast feature extraction algorithm, and yet it performed relatively poorly in comparison. Lukas-Kanade optical flow is generally considered a robust and efficient method for calculating the relative motion

Table 3.4: Image Alignment Performance Comparison on iPhone 5

	ORB/LK Optical Flow		Proposed Implementation	
X Offset (px)	**Features**	**Alignment**	**Features**	**Alignment**
-20	112ms	309ms	3.84ms	55.0µs
-10	107ms	318ms	3.62ms	54.2µs
0	112ms	310ms	3.60ms	55.1µs
10	107ms	314ms	3.77ms	54.6µs
20	108ms	322ms	3.60ms	53.9µs
	Error: $\pm 0.00005px$		Error: $\pm 0.05px$	

of feature points. Our results confirmed that it produces highly accurate alignment, but at a significant cost. In addition, while this implementation of optical flow has a fixed window size of 21px, our algorithm has no such limitation, provided the estimate is reasonable.

3.3.4 Translation Noise

We also look at how a poor estimate affects the quality of the computed alignment. This is particularly important as some of the biggest alignment issues are caused by errors in the gyroscope. The estimate is used directly as an input into the `FISA` algorithm and thus it is critical that we find the correct offset even if the input estimate contains significant error. We fix $dy = 10$ and the bin size is $2px$ for this evaluation, and use the same data set as described in Section 3.3.3.

Table 3.5: Translation with noise.

Input Error		Output Error (px)		
S.D.	**Samples**	**Mean**	**S.D.**	**S.E.**
0px	58.1	0.32	0.36	0.012
5px	58.1	0.32	0.36	0.012
10px	58.1	0.32	0.36	0.012
15px	58.1	0.32	0.36	0.012
20px	58.1	0.32	0.36	0.012

From the results in Table 3.5, we can see that despite significant error in the estimated translation, the alignment is still computed accurately. The results are identical each time which indicates that all features were matched in the same way.

3.4 Summary

We have described an efficient vertical edge alignment algorithm that combines both camera and sensor data for efficient operation. We have shown that it has excellent performance on current generation mobile devices, and that it is tolerant to both sensor and visual noise.

Next, we discuss the Fast Integral Sequence Alignment algorithm which is used to implement our tracking algorithm.

Chapter 4

Fast Integral Sequence Alignment

Finding the correspondence between two sequences of numbers can help us align images. We looked at statistical methods for computing the correlations between two sequences[79], and in particular, how this has been applied to computer vision problems in the past[80]. As a result, we developed a correlation function with a formulation that allows for efficient implementation, and we show how it performs directly on the iPhone 5.

The Fast Integral Sequence Alignment is essentially a peak matching algorithm. It works by looking at "peaks" in the given sequences and measuring how they match up. When the sequences are matching up sufficiently well with a given offset, such that the error between individual corresponding peaks is low, we say that the sequences are aligned. As an example, here, u and v are aligned with an offset of 3:

$$u = [3, 7, 8, 7, 6, 0, 0, 7, 5, 3] \tag{4.1}$$

$$v = [7, 7, 0, 0, 7, 5, 4, 0, 1, 5] \tag{4.2}$$

The definition is similar in principle to convolution, and practically speaking, the result is similar to FFT based image alignment[81]. However, FFT based approaches are generally not fast for small data sets, such as the ones we are dealing with, despite having a better computational efficiency[82].

In this chapter, we discuss the precise mathematical definition of this function, which is used in our proposed algorithm as discussed in Chapter 3.

4.1 Definition

Given two sequences of positive integers, u and v, of length n, the following sum of squared errors correlation:

$$(u * v)(k) = \sum_{i=0}^{n-1} [u(i) - v(i-k)]^2 \tag{4.3}$$

should yield minima when the two arrays u and v are aligned. In cases where $u(i)$ or $v(i-k)$ are undefined, the difference is 0. For a general pair of sequences, there is the potential that there are multiple values of k that produce good alignments, especially in the case that the arrays contain noise.

Error calculations are inherently based on the size of the overlap between u and v. Therefore ideally, $^{-n}/_2 < k < {}^{n}/_2$. Our confidence of a good match when k is outside these bounds is reduced, as the the minima will naturally be lower as less items overlap[1].

We adapt this function to include an initial estimation parameter e:

$$(u * v)(k, e) = (k - e)^E + \sum_{i=0}^{n-1} [u(i) - v(i-k)]^2 \tag{4.4}$$

where e is assumed to be close to the actual value of k. The exponent E in the bias should be adjusted so that it approximates a gaussian distribution on the same magnitude as the actual error. This can be pre-selected or computed dynamically; in our implementation we use 2 for efficiency.

This estimation bias serves two important purposes, it reduces the ambiguity in the case that there are multiple values of k that give a good alignment, and it provides several opportunities to improve the performance of the implementation. In practice, the estimation bias is how we leverage the sensor data - the more accurate the initial estimate, the more efficient and accurate the result.

[1] The mean squared error is another statistical formulation which effectively normalises the error by the size of the overlap, but makes our approach hard to optimise and doesn't make a significant difference in real usage.

4.1.1 Reducing Ambiguity

In the case that there are multiple minima, we need some way to distinguish between them. The initial estimate bias solves this by increasing the error for solutions further away from e. For example in the following,

$$u = [0, 5, 0, 4, 0, 5, 0, 4, 0, 5] \tag{4.5}$$

$$v = [4, 0, 5, 0, 4, 0, 5, 0, 5, 0] \tag{4.6}$$

an offset of $k = -3$ and $k = +1$ give the same minima 0. If we estimate that the alignment $e = 2$, we reduce this to a single minima at $k = 1$.

In the very rare case that we *still* have ambiguity, we can often pick the minima closest to e. However, in practical data sets, this event has never been observed.

4.1.2 Improving Performance

We can avoid evaluating $(u * v)(k, e)$ for values of k that are likely to give high error. We use an exponential initial error based on the distance of $k - e$ and if we incrementally evaluate $(u * v)(k, e)$, we can ignore values of k that are unlikely to yield good results.

The naive implementation has a best case $O(kn)$ because for all valid k, we must evaluate n multiplications and select the minimum. Leveraging the estimation, we can avoid computing k that are bigger than the currently found minimum.

We can improve the linear implementation by incrementally evaluating for k expanding outwards from e, avoiding computations for k that are bigger than the currently found minimum.

Alternatively, we could use a min-heap and incrementally evaluate the summation over i to track the current minima for a given offset k. In particular, it is common to find a good value for k within a few iterations. By specially crafting the heap `siftdown` function[83] we can leverage this condition to minimise the amount of cache thrashing.

Finally, in order to maximise the benefit of the heap, we should avoid evaluating i sequentially. Doing so will often compute many uninteresting cases where $[u(i) -$

$v(i-k)]^2$ is relatively small and cause the heap to reorganise itself repeatedly. In contrast, the biggest peaks will often lead to the biggest errors when data is misaligned. By evaluating these first, we push unlikely k to the bottom of the heap, and we reduce the chance that we continue to evaluate incorrect alignments. We compute the peaks of u such that $u[peaks[i]] >= u[peaks[i+1]]$ and use this to incrementally evaluate $[u(peaks[i]) - v(peaks[i]-k)]^2$.

4.2 Implementation

We define the function `FISA`, Fast Integral Sequence Alignment, for arrays u and v of length n as follows:

$$\texttt{FISA}(u, v, e) = \min_{k=-n/2}^{n/2} (u * v)(k, e) \tag{4.7}$$

This function, in practice, returns a $(k, error)$ pair such that error is minimised. The range of k can be adjusted depending on the data sets we are dealing with. We implement this function in C++11 with two main variations, a linear search method that computes for all k but is bounded by the worst error found thus far, and a heap method that incrementally updates based on the lowest error.

4.3 Evaluation

We evaluated the above algorithm (see Appendix A for an overview of the implementation) on an iPhone 5 running iOS 7, compiled with `clang++-3.3 -O3`. We vary the size of n using randomly generated sequences with values between 0 and 50. We add several large values in the data set between 0 and 250 and include up to 10% noise in all values. We compare 4 variations of the algorithm, including a linear search from left to right, the linear search expanding outwards from the initial estimate, the heap search with incremental evaluation and the heap search with peak-order evaluation. See Table 4.1 for the results.

For small problems $n < 256$, the outward expanding linear scan is the most efficient choice. This is likely due to the fact that it uses the L1 cache more effectively than

Table 4.1: FISA Performance Results.

	Linear (µs)		Heap (µs)	
n	**Left-Right**	**Outward**	**Left-Right**	**Peaks**
8	0.552	**0.484**	3.628	6.629
16	1.716	**1.357**	10.721	10.489
32	4.914	**2.878**	22.536	20.126
64	16.560	**9.496**	46.687	34.780
128	56.323	**32.793**	94.913	60.527
256	176.742	112.915	201.484	**103.439**
512	733.457	430.844	492.797	**205.429**
1024	2987.51	1713.51	1253.59	**365.713**

the heap implementation which copies a significant amount of data during `siftup` and `siftdown` operations.

The cost of the initial sort in the heap+peaks implementation is moderate but pays off significantly for $n >= 512$. For $n = 1024$, it is almost 35 times faster.

These results suggest that for a general algorithm, a hybrid between the two algorithms would be appropriate, however in our case we are mostly concerned with $n < 256$ so we have chosen to use the linear+outward search.

4.4 Summary

We have formally defined a new mathematical method for computing the offset between two integral sequences. We described four implementations and analysed the performance characteristics on current consumer level hardware.

Next, we discuss the Transform Flow toolkit which is used to develop and test our proposed Vertical Edge Alignment algorithm.

Chapter 5

Transform Flow Framework

Evaluating and comparing algorithms is an important part of quality research. The ability to compare different approaches with the same data sets is critical to the development of improved methods[70]. This is also true for mobile AR, where many tracking algorithms are developed specifically within the constraints of a particular hardware configuration and testing methodology.

Ideally, sensor and visual data, as outlined in Figure 1.2, is captured from a variety of mobile devices in such a way that it can be repeatedly replayed. Algorithms can be critically analysed, frame by frame, to isolate edge cases and bugs in a controlled environment; test cases can be developed and run automatically to check for regressions. In addition, with the right design, both offline and online processing of input data should be possible, so that algorithms can be seamlessly deployed on real hardware for user evaluation and application development.

An open source platform that facilitates the analysis of new and existing algorithms would allow researchers to develop and test new ideas easily. Existing code can be reused, which typically reduces the challenges and risks associated with software development. Such a platform could also serve as a useful educational tool for students wanting to learn about different tracking algorithms and how they are implemented.

In this chapter, we discuss the various components that make up the Transform Flow Framework, and how they have been used to support both the design and the evaluation of the the Fast Vertical Edge Alignment algorithm as discussed in Chapter 3 and Chapter 6 respectively.

5.1 Transform Flow Framework

We developed the Transform Flow Framework[1]. It includes data capture applications for iOS and Android, data visualisation and analysis tools, an abstract motion model interface for implementing algorithms, several sample implementations of different tracking algorithms, a browser application for running these tracking algorithms on supported mobile devices, and several data sets which we developed for testing and evaluation.

We have decided to publish the code, documentation, and data sets as completely unencumbered open source, under the MIT License, in the hopes that its availability will reduce the barriers for other researchers entering this field. In addition, we believe that the structure of our contribution can encourage a systematic approach to the development of new algorithms. The existing code base includes many practical examples, and over time we hope that our tooling can serve as a useful platform for a wide variety of tracking implementations.

This is an ongoing project and we realise that it may not fit everyone's requirements. We have specifically chosen online systems that support collaboration, so as to maximise the value of the toolkit - not just for our needs - but for the needs of this field in general. Transform Flow is a platform that encourages collaborative research, development and education in the area of mobile AR.

5.2 Motion Models

A motion model is an abstract interface (see Appendix B) that wraps the implementation of a specific tracking algorithm. The set of inputs and outputs are well defined, and relate specifically to the tracking task being performed. Multiple algorithms can be implemented with the same programming interface, which allows the visualisation and browser applications to interact with different implementations without significant changes.

We specifically designed our motion model abstraction around the hardware capabilities of modern mobile devices and the types of data required for accurate global outdoor AR tracking. Specific types of motion models may require additional inputs

[1] Available online: https://github.com/HITLabNZ/transform-flow

or outputs. For local image based tracking, we may output a camera pose relative to a local frame of reference. Other motion models might require direct access to the magnetometer. This is supported by subclassing and modifying the source code appropriately.

To serve as an example, we include two motion models in the current distribution. These motion models can be compiled and used both in the visualisation tool and in the browser application:

5.2.1 Basic Sensor Motion Model

The included basic sensor motion model implements a sensor fusion based tracking algorithm. It combines the compass and gyroscope using a low-pass filter for improved bearing calculations. It tracks relative changes in rotation around the gravity axis, and exposes these as an output.

The bearing is typically measured as the angle between a device coordinate frame and true north. The device usually defines a fixed axis, by default on iOS devices it is +Y. Because of this, rotations around the camera axis -Z may cause changes in the bearing (see Figure 3.2). Latency in sensor fusion can create significant tracking artefacts, such as temporary instability in the bearing output. To compensate for this, we project the bearing into a global coordinate system and then measure the rotation of the camera axis relative to North. This reprojection ensures that rotations around the camera axis do not elicit changes in bearing.

In order to initialise the basic sensor motion model, at least one motion update is required to establish a gravity vector, and one heading update to establish a global bearing. After that point, gyroscope updates are combined with heading, and along with the gravity vector and GPS, this provides a fairly robust global frame of reference.

5.2.2 Hybrid Motion Model

The hybrid motion model derives from the basic sensor motion model and incorporates image processing into the bearing calculation. It implements the approach discussed in Chapter 3 and Chapter 4, and uses the rotation about gravity as the estimate for image processing. In the case of a visual tracking failure, sensor fusion

provides continuity but typically at a decreased level of accuracy. This ensures a robust output even in challenging scenarios.

5.3 Data Capture

We created a mobile data acquisition system[84] to capture sensor data and video frames. This application was developed for iOS using Objective-C. The rate at which sensor data and video frames are captured is independent, and can be customised at compile time. All data is saved in a CSV (comma separated values) formatted log file, including video frames that reference external PNG (portable network graphics) image files in the same directory.

Sensor data is captured using Apple's CoreLocation and CoreMotion frameworks. CoreLocation provides WGS84 latitude, longitude, altitude and bearing, while CoreMotion provides gravity, linear acceleration rotation rate and magnetic flux. The position and bearing typically have quite a high latency, while the motion data is usually captured at 30Hz.

Video frames are captured using AVFoundation that provides access to the camera at a variety of resolutions. We record the captured image as interpolated RGB, which is universally supported across all devices and a convenient format for further processing. We typically capture at 480×360 at 10Hz - higher frame rates generate huge amounts of data and can be harder to analyse.

Phone specific data including the device name and hardware identification are recorded as one of the first entries in the log file. This can be used for hardware or device specific calibration files.

In order to support global tracking algorithms, when recording is triggered, the current position and bearing are written to the log file if possible. This allows at least one global position and bearing entry to exist before frame data and motion data, which is typically required for initialisation.

The tool includes a real-time visualisation of sensor measurements. This allows for a greater understanding of the effects that the physical device motion is having on the sensor data. It can be instructive to check the gyroscope, accelerometer or other sensors while manipulating the device. Per-device issues including calibration problems and drift can be assessed quickly and isolated.

FIGURE 5.1: The data capture application running on an iPhone 5.

5.3.1 Android Support

An existing Android data capture tool, developed by another researcher, Alexander Pacha, has been modified and contributed to the Transform Flow project[85]. It can output Transform Flow style data sets from a wide range of Android devices. It has not yet been extensively tested, but supporting a wide range of hardware can be challenging. The coordinate systems for inertial sensors may not be consistent, the field of view for the camera may not be correct, the sensor output may be more or less accurate than generally expected, and the latency in sensor measurements might be significantly different from the norm. Manufacturer and platform specific implementations of the sensor fusion algorithms may also produce different results.

Dealing with these issues is critical for wide-spread deployment on consumer grade hardware. The Android data capture tool is the first step to understanding and solving these problems, and this process will present many good opportunity to further refine and collaborate on the Transform Flow library.

5.3.2 Data Set Format

Data sets are recorded using the on-screen switch. This information is saved into the phone's memory and can be downloaded to a computer using the Xcode organiser. The data sets themselves were designed to be simple to work with and flexible enough to support future requirements.

We use CSV as it is a simple format for structured data and can be logged easily; a sample is shown in Figure 5.2. We uniquely identify each row in the log using an

index starting from 1, a function name (e.g. "Gyroscope") and a timestamp. The function name relates to the structure of the remaining arguments in the row and is used for processing rows into useful data structures (refer to Appendix C for a more detailed specification).

```
1, Location, 349238.3985, -43.521526, 172.582418, 21.4788,
   10.0000, 16.0000
2, Heading, 349238.5007, 80.6878, 104.2185
3, Gyroscope, 349239.1022, -0.034565, -0.038651, -0.003393
4, Accelerometer, 349239.1022, 0.000856, -0.008263, -0.015212
5, Gravity, 349239.1022, -0.000902, -0.998390, -0.056718
6, Motion, 349239.1022
7, Frame, 349239.1090, 0
```

FIGURE 5.2: A few lines from a data set log file.

It is easy to modify the data capture tool to add additional data structure records, e.g. adding battery status, GPS course, current waypoint, or other parameters that may be of interest. This allows for the data capture application to be extended with new capabilities as required for specific research areas, while retaining core functionality and compatibility with existing processing/visualisation tools.

Nested data can be supported using sequential relationships. For ease of readability, gyroscope, accelerometer and gravity vectors are recorded separately, but are immediately followed by a motion entry with the same timestamp, which ties them together into one logical unit for processing. In the future, additional motion specific fields could be added, e.g. magnetometer measurements, without modifying any other structures.

5.4 Visualisation

We have developed a desktop application[86], shown in Figure 5.3, for viewing data sets captured using the mobile data acquisition tool. This application is written in C++11 and uses OpenGL for rendering. It currently compiles for Mac OS X and Linux support is almost complete.

Our current implementation assumes that the user is interested in visualising data within a global frame of reference, with position defined by (latitude, longitude, altitude) and rotation defined by (bearing, gravity). To systematically capture this

data, we define an abstract motion model that is used to process incoming sensor data and image frames.

The camera pose is computed in two steps, a quaternion rotation based on the bearing and gravity information, and a displacement based on the latitude and longitude. We use East, North, Up (ENU, maps to XYZ) Cartesian coordinates which are intuitive and practical for the small data sets we are usually evaluating.

The input data set, combined with a motion model, produces an immutable per-frame globally registered camera pose. This sequence of frames can then be visualised and analysed without further motion processing. In the case that a motion model requires several sensor updates before it can be initialised reliably, the visualisation will start at the first frame after the localisation becomes valid.

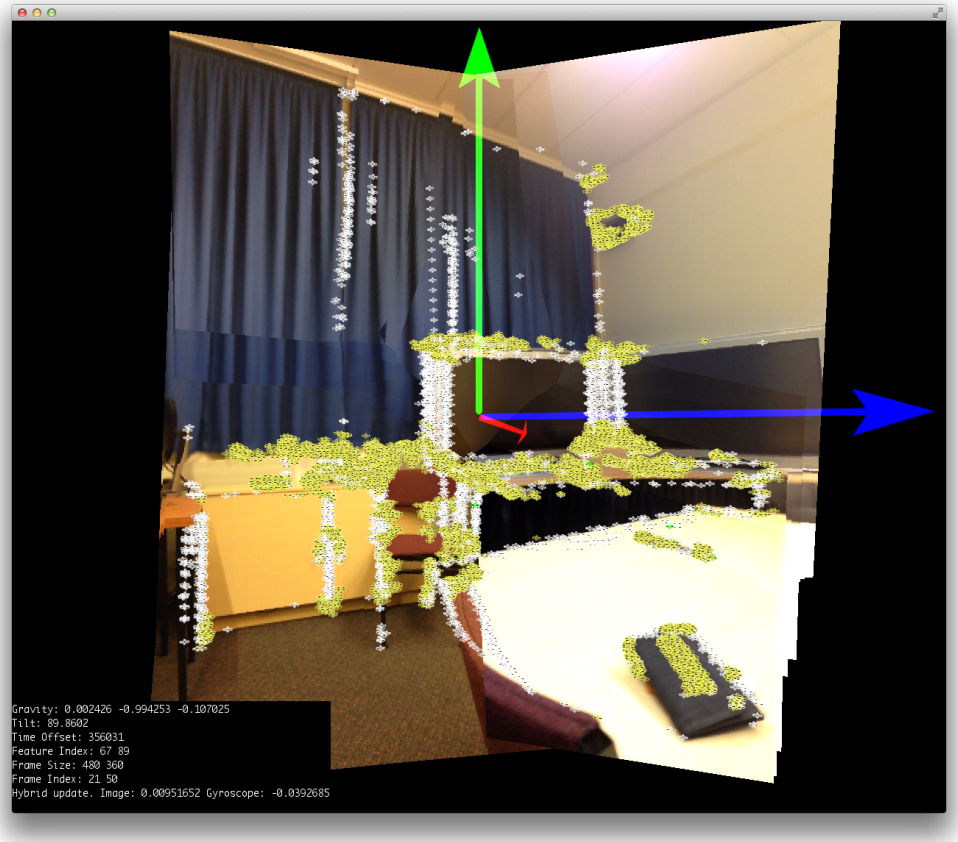

FIGURE 5.3: The Transform Flow Visualisation application rendering 50 combined frames in a 3D environment. The makers have been added by different feature detection algorithms.

Displaying frames in a 3D environment is challenging because a camera frame is a planar projection of a typically non-planar environment. Without depth information it is impossible to accurately reproduce the actual 3D structure. To work around

this problem, the visualisation tool uses planar projections based on the camera's field of view and an arbitrary scale factor. Frames can then be rendered in a 3D environment and explored using an arbitrary view position. Pure rotations result in the easiest to interpret visualisation as there is no change in planar alignment.

5.4.1 Analysis

Visual inspection of data sets in a 3D frame of reference (see Figure 5.4) provides a good high level overview of algorithm behaviour and exposes tracking errors including incorrect coordinate frames (e.g. rotation on the wrong axis), sensor integration issues (e.g. large jumps between frames) or bad feature point detection.

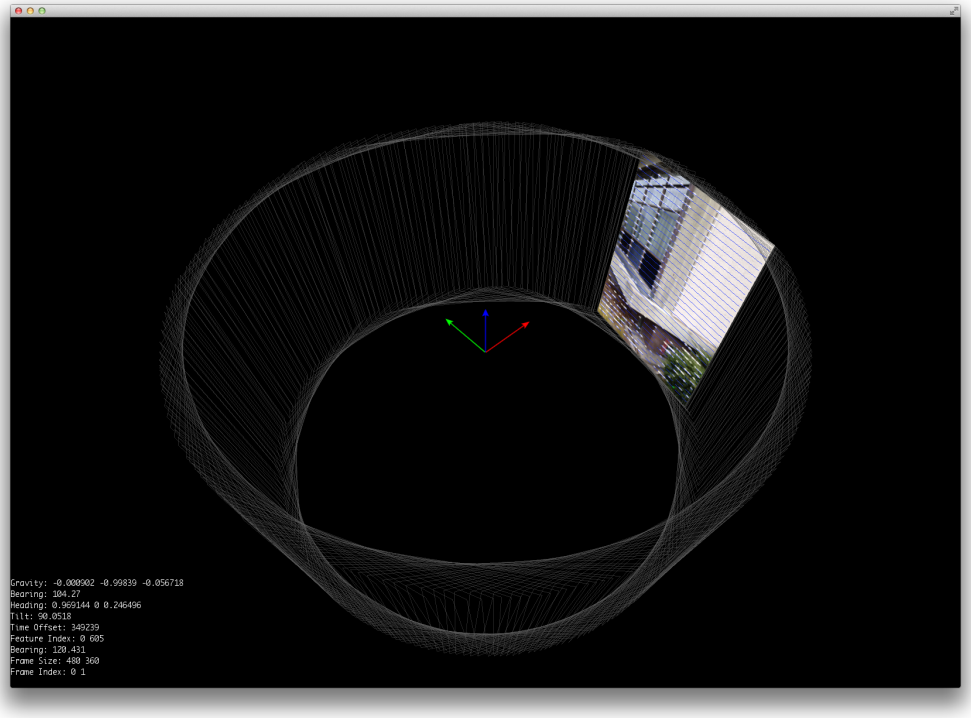

FIGURE 5.4: A 360° panoramic data set, with the first frame visible, in the Transform Flow Visualisation tool. The user can interactively navigate through individual frames and view their associated metadata.

Each frame can have a set of debug notes and visual markers associated with it. These are essentially debug messages from the motion model. Per-frame logging is a useful debugging tool and provides specific feedback about how the motion model was working internally. Per-frame feature points show a structured analysis of the image data and allow a user analyse the details of individual points by selecting them visually.

5.4.2 Evaluation

Systematic evaluation is an important process for ascribing a metric to the quality of a tracking algorithm without manual intervention. It allows for the critical and consistent comparison of different algorithms over a wide variety of data sets and for the comparison of the same algorithm as it is being developed and tweaked.

Transform Flow includes support for tracking points, which are spatial markers registered against pixel coordinates for a specific image frame in a specific data set. Tracking points are stored in a separate CSV file inside the data set, and the visualisation tool provides some features to assist with the generation of this data. Tracking points which represent the same visual feature are grouped by a tracking index number, and these form the basic data structure required for further processing.

We have implemented a method to evaluate the accuracy of per-frame bearing. We use groups of static tracking points (such as in data set 2013D0 shown in Figure 5.5) which are generally fixed physical features, and compute the 3D position of these points using the camera pose computed by the tracking algorithm. We project these points on the plane $Z = 0$ and compute the bearing. With purely rotational motion, the relative bearing should not change, which is our ground truth. We compute the bearing of each tracking point and measure the average, standard deviation and standard error. The stability of this metric reflects the quality of the tracking algorithm.

FIGURE 5.5: The 2013D0 tracking point shown in four frames.

In the future, we would like to add a projective based evaluation technique[87]. Given the same feature point in multiple frames, for all frames which contain that feature point, the projection of feature point into a 3D environment should converge to a single 3D position. The distribution of this convergence is directly related to the

quality of the tracking algorithm, and similarly to the bearing evaluation, provides us with a metric to compare different approaches systematically.

5.4.3 Sample Data Sets

In order to facilitate consistent testing and evaluation, and additionally provide sample data for the visualisation tool, 14 data sets have been published and documented[88]. These data sets are captured using the published iOS capture tool, and include a full range of sensor measurements. We hope that the publication of these data sets will stimulate others to do the same and that over time a large corpus of sample data can be built up to support a wide range of evaluation techniques.

5.5 Deployment

We have developed an iOS application[89] that can be used to run algorithms developed using the Transform Flow motion model abstraction. It uses a similar setup to the capture tool, but rather than logging the events, it applies them directly to a motion model. The AR visualisation is then rendered using the calculated frame of reference.

The application itself consists of two parts, a map view and an AR browser (see Figure 5.7).

The map view shows the user position and nearby globally registered points of interest. It provides an overview of where content is located and how it relates to local buildings and geography. The built in map view overlays the GPS accuracy and gives a useful indication as to the tracking quality.

The browser can display points of interest using 3D content (via Wavefront OBJ files) and 2D billboards (constructed directly from UIView instances), which are rendered on top of a real-time video stream using OpenGLES. The visualisation includes a planar grid which is useful for understanding the practical tracking quality, e.g. jitter, rotations. The implementation itself is multi-threaded, and uses Grand Central Dispatch to offload the rendering, camera frame capture and motion model computations to separate CPU cores so that the main user interface remains responsive.

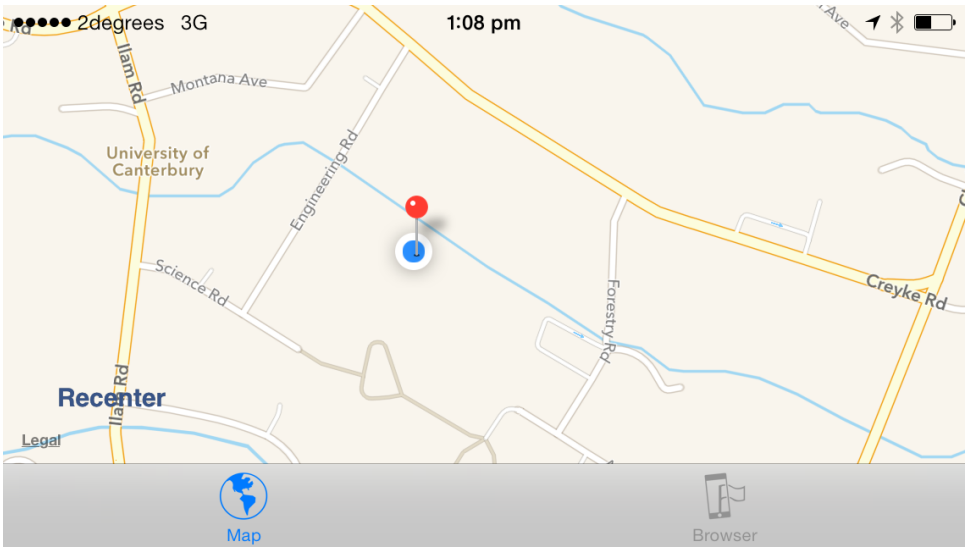

FIGURE 5.6: The Transform Flow Browser, showing the top down map view, with a single marker.

FIGURE 5.7: The Transform Flow Browser, showing a 3D model on top of The HIT Lab NZ.

Dealing with a wide variety of camera and screen configurations is not trivial. Specifically, different cameras have different intrinsic properties, the most important being the field of view. We estimate this parameter as 55° which is, in practice, ±2° for commonly used iOS devices. Device specific calibrations might be useful in a research setting but would be cumbersome for end user applications. Ideally, a per-device database of camera intrinsic properties would allow for wide spread deployment with acceptable accuracy, or for unknown devices, some kind of online calibration procedure.

5.5.1 Android Support

The browser application is very much platform specific code as it involves custom UI, rendering and other functionality. In addition, custom code is required for interacting with the hardware (e.g. cameras, sensors). This means that there is a moderate amount of per-platform work required. However, the Transform Flow library and it's supporting libraries all support cross-platform compilation. Because of that, the core tracking functionality need not be reimplemented for different platforms.

Several researchers are presently working on adapting the existing AndroidAR browser to support Transform Flow via the Android NDK. We hope this work progresses and allows us to support a wide range of devices.

5.6 Source Code

One of the important goals of this project is to create something that will facilitate collaboration and further research. As such, the data capture, analysis and browser tools have been released under the MIT license on GitHub. The MIT license allows developers to use the source code free of charge with very few limitations (e.g. commercial use is acceptable).

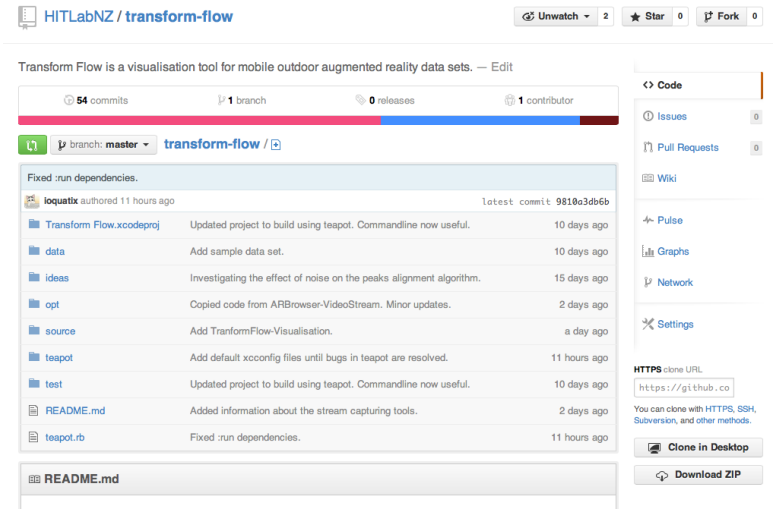

FIGURE 5.8: The Transform Flow repository on GitHub.

GitHub provides a fantastic environment based on Git[90] where other researchers can easily fork the source code and contribute back their modifications. We hope to

integrate new motion models and evaluation methodologies as they are developed, so that our project can serve as a useful tool for comparative analysis for future work.

5.6.1 Building

Existing research[91] and practical experience has shown that build related challenges can be a major source of problems:

- Choosing appropriate compilers and SDKs for the target platform.
- Selecting compiler flags in a way that is consistent across all systems.
- Specifying the directories where headers and libraries from other required software packages are located.
- Generating code and copying resources, e.g. shaders and sample data.
- Specifying the location(s) to produce object code, libraries, executables and install packages.

Our project spans approximately 30 individual libraries of various scale, as outlined in Section 5.7. We are required to build many of these packages in different environments, e.g. Mac, Linux, iOS, Android, with different compilers and different system configurations. In addition, we wish to encourage collaboration, so the process of building, running and extending the software should be as easy as possible, despite the inherent incompatibilities in the supported platforms.

Originally we were using CMake[92] to build several libraries. As the project got larger and included more dependencies, the amount of CMake configuration required became unmaintainable (120+ lines of configuration per logical package spread over 6 configuration files by default[93]). In addition, CMake doesn't provide any facilities for pulling in versioned dependencies which means that users must install these manually for all required platforms, which is generally a very complex and error-prone process.

We have noticed that as you increase the cost of building and managing software (i.e. non-code related work, external dependencies, build systems), you proportionally increase the tendency for developers to combine disparate functionality (i.e.

threading, networking, image processing, audio) into a single project. By doing so, the ratio of productive code to unproductive configuration is improved which generally reduces the burden of developing and maintaining the library. We refer to this as monolithic project feature creep.

The problem with this approach is that in many cases, you may only require a small subset of the total functionality. Large projects have many drawbacks:

- Scope of project is beyond any single person to manage and hard for new developers to comprehend.
- Compatibility and support across multiple platforms and releases is difficult.
- All or nothing approach to individual components, no way to upgrade/version only specific parts.
- External dependencies left up to the user or included directly, integration and build is complex.

To mitigate these issues somewhat, projects often include complex configuration systems that in our experience, frequently break on cross-platform builds or unanticipated configurations (this was a major source of problems earlier in the research). The end result is that it is practically impossible to use these large libraries of code while still ensuring that the system is easy to build and extend.

There are many good examples of "large projects". OpenCV includes in it's own source tree, versioned copies of `libjpeg`, `libpng`, `libtiff`, `zlib`, `jasper`, `openexr`, and `ffmpeg`. Up until recently, it was impossible to build some parts of OpenCV for iOS. This is not uncommon: `boost` includes includes over 50 separate components in a single package, and up until recently did not support iOS or Android without custom modifications to the source tree and build system. Because the library follows a monolithic release schedule, fixes were not available for a significant period of time.

We believe that small modular packages are better: they integrate with existing projects more easily, they are simpler to understand and debug, they are less daunting to fork and modify, they can be tested in isolation, they minimise the amount of unnecessary code in a project, and most importantly they make it possible for designs to evolve in a localised fashion[94]. However, there are no existing systems

which are generally designed to support modular C++ code where cross-platform builds are the norm, so we developed one.

5.6.1.1 Project Growth

The Transform Flow Visualisation application was developed throughout the duration of the research, and was not split apart into separate components until we were required to do so to support the browser application. Over the course of the research, it gained many new features, including tracking algorithms, particle rendering, image billboarding, sensor processing, and so on. This is a classic example of monolithic project feature creep, and made it impossible to port the software directly to iOS.

Only once the core tracking and sensor fusion processing algorithms were extracted into a separate library and isolated from the concerns of the visualisation tool, was it possible to build and deploy our algorithms on mobile devices. We actually always intended to split the project up like this, but lacked the specific tools to support this development methodology, which is why we developed Teapot.

5.6.1.2 Teapot

Teapot[95] is decentralised package manager and build tool that can fetch and compile versioned dependencies for multiple target platforms. It includes minimal task specific code and instead delegates this to external packages where possible. The inspiration for this tool came from RubyGems[96], which is essentially a platform-independent package management system with very minimal per-package overhead (a single file, about 20 lines). Similarly, Teapot requires only a single file per project by default, which typically includes a list of external dependencies, and a set of targets which can be built.

Teapot is used for all parts of Transform Flow and can build any and all parts of Transform Flow across all supported platforms with minimal configuration by the user. It uses a package metaphor as a central structure for managing the build process, where individual packages are considered immutable units of source code. Abstract dependencies, i.e. dependencies which can be satisfied by more than one package, are the primary method to customise the build process. As shown in Section 5.7.2, Teapot can dynamically reconfigure the build depending on the

specific requirements of the user and platform, while maintaining the consistency guarantees afforded by individual packages.

5.6.2 Unit Testing

Ensuring that changes and modifications don't introduce problems with existing code is a useful requirement for collaborative projects. GitHub, when combined with Travis-CI[97], provides immediate feedback when users make source code submissions (pull requests) on public projects.

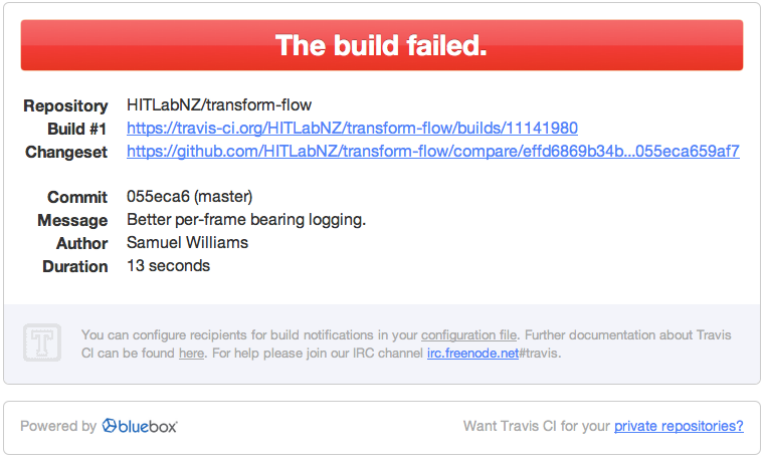

FIGURE 5.9: An email notification from Travis-CI due to a build issue.

Many component parts of the Transform Flow toolkit have unit tests, which means that any time the code is changed, either directly in the repository on GitHub, or as a pull request from another user, the unit tests will be run and the results reported appropriately. This helps to ensure that new users making contributions can feel confident that they are not breaking existing code and supports existing maintainers who might be refactoring or adding new features.

In addition, unit tests also serve to document various parts of the system and how to use them. Functional tests provide useful examples of specific functionality and integration tests show how to combine different parts of a system. This allows new users to become familiar with the code quickly and easily, and serves as a working example of the available functionality and how it should be used.

5.7 Software Overview

The Transform Flow Evaluation and Visualisation Toolkit is a collection of libraries, data sets and applications to support the development of mobile AR tracking and applications. It includes C++, Objective-C and Java code, spread over several repositories:

- Transform Flow
  https://github.com/HITLabNZ/transform-flow
- Transform Flow Capture for iOS
  https://github.com/HITLabNZ/transform-flow-capture-ios
- Transform Flow Capture for Android
  https://github.com/HITLabNZ/transform-flow-capture-android
- Transform Flow Data Sets
  https://github.com/HITLabNZ/transform-flow-data
- Transform Flow Visualisation
  https://github.com/HITLabNZ/transform-flow-visualisation
- Transform Flow Browser for iOS
  https://github.com/HITLabNZ/transform-flow-browser-ios
- Transform Flow Browser for Android
  https://github.com/HITLabNZ/transform-flow-browser-android

5.7.1 Compatibility

Table 5.1: Transform Flow Compatibility.

Component	iOS	Android	Mac	Linux
Core Libraries	Supported	Supported	Supported	Supported
Capture Tool	Supported	90% Done	N/A	N/A
Visualisation Tool	N/A	N/A	Supported	In Progress
Browser	Supported	10% Done	N/A	N/A

5.7.2 Transform Flow Structure

The Transform Flow Visualisation software includes many modules. These modules implement specific areas, such as the core algorithms, the visualisation primitives, image loading, etc. Depending on what high level part of Transform Flow is being build, different modules are required.

The visualisation tool uses the Dream framework[3], and thus pulls in many dependencies specifically for rendering and OpenGL, as shown in Figure 5.10.

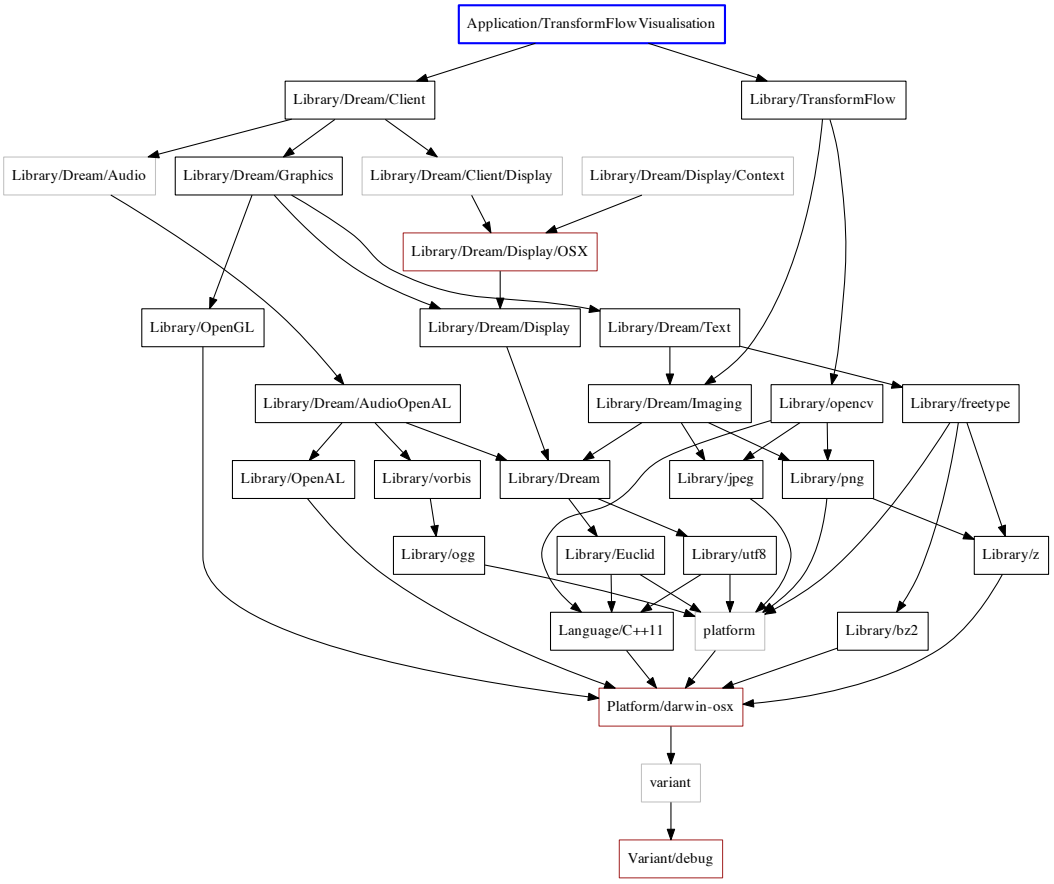

FIGURE 5.10: The structure of the Transform Flow Visualisation application, generated by Teapot, for Mac OS X.

In comparison, while the Transform Flow Browser application still depends on the core Transform Flow algorithms, it doesn't require the same rendering infrastructure, as shown in Figure 5.11.

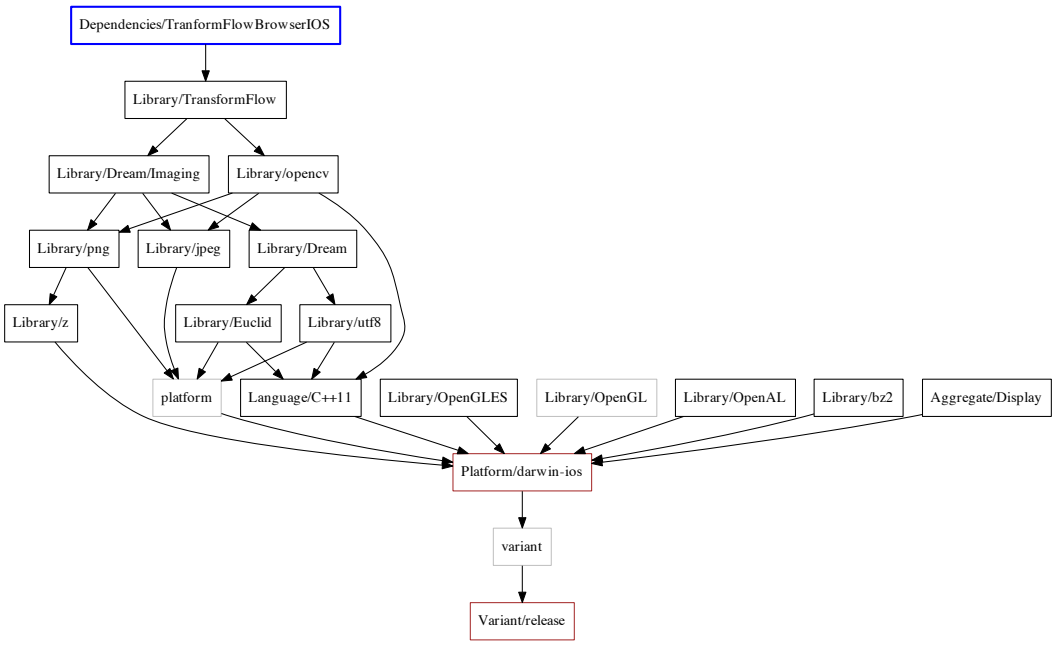

FIGURE 5.11: The structure of the Transform Flow Browser for iOS, generated by Teapot.

5.8 Summary

We have discussed the development of an open source toolkit for mobile AR development, including algorithm implementation, data capture, visualisation, analysis and deployment on real hardware. We have explained how this can be used for systematic evaluation, and published a number of data sets for fair comparative analysis. Our implementation is designed from the ground up for cross platform compatibility and has been structured to support future research in this area. This toolkit is critical for the on-going evaluation of mobile AR, and is the first publicly available system for developing and quantifiably evaluating mobile AR tracking algorithms.

Next, we evaluate our proposed algorithm to show how it works in real world conditions.

Chapter 6

Practical Evaluation

Current outdoor AR platforms provide varying levels of responsiveness. Imperfections in the visual alignment and poor responsiveness to changes in device orientation may cause frustration and confusion. We evaluate our algorithm with real world data sets and on real world devices to find out how our proposed algorithm affects accuracy and tracking performance, and whether this is an overall improvement for the usability of AR applications.

In this chapter, we test our proposed algorithm as discussed in Chapter 3 and Chapter 4 in a variety of real world situations using the tools discussed in Chapter 5.

6.1 Real World Accuracy

We collected several data sets to support the evaluation of our algorithm. We focused on environments containing at least several vertical edges, as this was one of the key assumptions for our tracking algorithm to function effectively. We evaluated data sets that include motion blur, moving objects (cars, people), combinations of foliage and buildings and other typical situations in mobile outdoor AR.

We use the Transform Flow Visualisation and Evaluation Toolkit[86] as discussed in Chapter 5 for measuring the quality of the tracking algorithms. We analysed the bearing of fixed tracking points in several data sets and compared the relative accuracy of the sensor fusion tracking algorithm in comparison to our proposed hybrid tracking algorithm.

6.1.1 Results

We present results from four data sets in detail (sample frames shown in Figure 3.6, Figure 3.7, Figure 3.8, and Figure 3.9). The tracking points used for evaluation were all on static vertical edges and thus we expect the deviation should be close to 0° with a perfect alignment.

TABLE 6.1: Relative Bearing Accuracy

	Sensor Tracking		Hybrid Tracking		
Data Set	**S.D.**	**S.E.**	**S.D.**	**S.E.**	**Improvement**
2013A0	0.37°	0.08°	0.29°	0.06°	1.3×
2013A3	8.88°	1.81°	0.35°	0.07°	25.4×
2013A4	2.82°	0.54°	0.64°	0.12°	4.4×
2013A5	5.93°	1.12°	0.85°	0.16°	7.0×
2013B0	2.65°	0.38°	0.76°	0.11°	3.5×
2013B1	2.24°	0.26°	2.22°	0.25°	Negligible
2013B2	9.52°	1.47°	1.70°	0.26°	5.6×
2013C0	2.93°	0.49°	0.69°	0.12°	4.24×
2013C1	2.55°	0.43°	0.87°	0.15°	2.9×
2013C2	8.34°	1.74°	7.98°	1.66°	Negligible
2013D0	1.17°	0.28°	0.88°	0.21°	1.3×
2013D1	1.83°	0.47°	0.48°	0.12°	3.8×
2013D2	1.29°	0.30°	0.18°	0.04°	7.2×

From these results we can see that the image based alignment algorithm in almost all cases makes a significant improvement. Several notable examples include 2013A3, 2013A5 and 2013B2. In these cases, errors in the compass and gyroscope cause massive visual shifts such as in Figure 6.1.

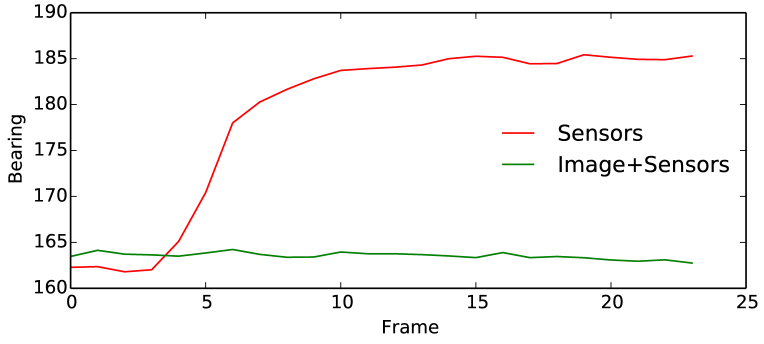

FIGURE 6.1: Bearings computed for 2013A3, showing a significant errors in the sensor data.

Other difficult situations include fast motion and motion blur, such as in Figure 6.2. Our feature point extraction works reliably even in these difficult conditions.

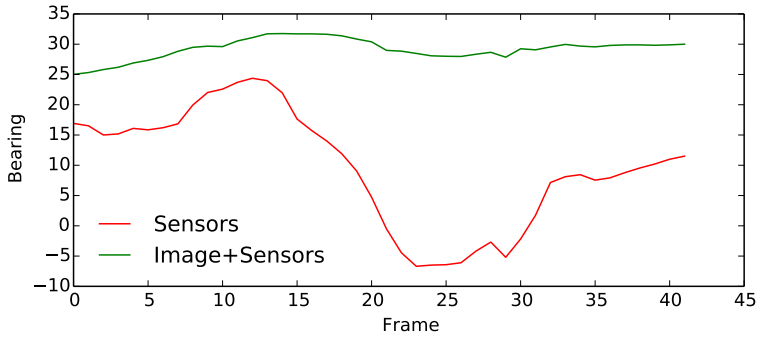

FIGURE 6.2: Bearings computed for 2013B2, showing a significant errors in the sensor data.

In some cases, there is little improvement. In the case of 2013C2, there is significant motion blur combined with few vertical edges which reduces visual alignment confidence which causes the algorithm to revert back to sensor tracking. This is expected behaviour but in effect reflects a situation where there was not enough visual information to improve the alignment within our current statistical model.

6.2 Real World Performance

We profiled the browser implementation and analysed the cost of image processing based on frame size and scanline width dy on an iPhone 5 attached directly to a computer (i.e. not running on battery power). The image processing time was measured for 600 frames captured at 30Hz while the phone was statically positioned looking at a scene of moderate complexity. We made the measurement twice and discarded the first set of results.

6.2.1 Results

We evaluated 3 different resolutions, 480×360, 640×480 and 1280×720 and various appropriate values for dy.

The results in Table 6.2 show that we have adequate options to process images in real time on current generation mobile hardware, even at 1280×720. However, at this resolution, memory contention and bandwidth in general became the biggest performance problem. The overhead costs of the higher resolutions are significant,

Table 6.2: Real World Performance Comparison

		Frame Time				
Resolution	**dy** (px)	**Mean**	**S.D.**	**S.E.**	**Max**	**FPS**
480 × 360	5	14.9ms	2.9ms	0.1ms	24.8ms	67.1
480 × 360	10	5.7ms	3.5ms	0.1ms	15.7ms	177.0
480 × 360	15	2.9ms	2.4ms	0.1ms	11.8ms	344.7
480 × 360	20	1.9ms	2.0ms	0.1ms	9.6ms	515.0
640 × 480	10	16.0ms	3.2ms	0.1ms	24.4ms	62.5
640 × 480	15	11.4ms	4.1ms	0.2ms	21.4ms	87.9
640 × 480	20	8.6ms	4.3ms	0.2ms	17.5ms	115.8
640 × 480	30	5.4ms	3.6ms	0.1ms	13.7ms	183.7
1280 × 720	20	29.1ms	3.6ms	0.1ms	38.2ms	34.3
1280 × 720	30	20.6ms	4.0ms	0.2ms	30.2ms	48.5
1280 × 720	40	16.0ms	3.9ms	0.2ms	26.2ms	62.3
1280 × 720	80	8.8ms	4.0ms	0.2ms	21.7ms	114.1

e.g. copying the image buffer was the 2nd largest cost associated after image processing itself, both from the camera device to main memory and from main memory to OpenGL for the video background.

Despite the 1280 × 720 resolution only being 5× as much data, the performance was 15× worse. We theorise that this is due to the performance of the CPU cache. 480 × 360 is approximately 500Kbytes of RGB data (3 bytes per pixel), which is about half the available CPU cache on the Apple A6 CPU powering the iPhone 5. The 1280 × 720 video frame is approximately 2.6Mbytes, which is significantly more than can fit in the CPU cache. The scanline algorithm implementation is not cache aware, and thus may cause many L2 cache misses in practice.

In addition, due to the multi-threaded design of the browser application, the full performance of the phone may be diminished in comparison to synthetic benchmarks which run exclusively. The CPU and GPU memory is shared and thus the total throughput will be limited depending on the tasks being performed on other threads. The benefits of two independent CPU cores may be diminished if the processing is memory intensive.

For higher frame resolutions, the temperature of the phone increased significantly, yet the practical usability of the browser was not significantly altered. The battery life and physical usability of the phone were likely diminished. For real-time augmented reality, we found 480 × 360 had sufficient image quality with few of the physical discomforts associated with the higher resolutions.

Accurate field of view is required for correct image based tracking. While Android provides a way to access this information programatically, it has to be hard-coded on iOS devices. While we could measure the performance of processing frames at 1280×720, we could not use the resolution in practice as it has a different aspect ratio and field of view. An appropriate bug report has been filed.

6.3 User Study

A user study was designed to evaluate whether improved tracking would help users to engage with AR content more precisely. We focused on a task that would require accurate physical control to align the mobile device with a globally registered object. By looking at task completion time, we can measure whether different tracking algorithms have a measurable effect on the users' ability to complete the task, and ideally assert that improved tracking can benefit practical AR interactions.

6.3.1 Task

The AR browser application was modified into a small game where users were required to centre a reticle over a virtual target. One virtual target was placed around the user at a distance of approximately 35m. The user was required to hold the reticle over the target accurately for 0.5s to trigger the completion of the task. After 1s, a new target would be placed somewhere in the world, and the user could repeat the process.

In order to test the influence of the different algorithms on the user response time, two modes were used for the tracking task. In the 1^{st} mode, sensor fusion is used for tracking, while in the 2^{nd} mode, the hybrid tracking algorithm is used. To elicit feedback from the user, in the sensor fusion mode, a red target is used, while in the hybrid mode, a blue target is used.

The time taken for the task was measured starting from when the reticle intersects the outside of the target, to when it has been held accurately in the centre for 0.5s. If the user moved sufficiently far outside the targeting area, the time would be reset and they could attempt the task again with no penalty. This method ensures that the time it takes for the user to find the target is not part of the evaluation.

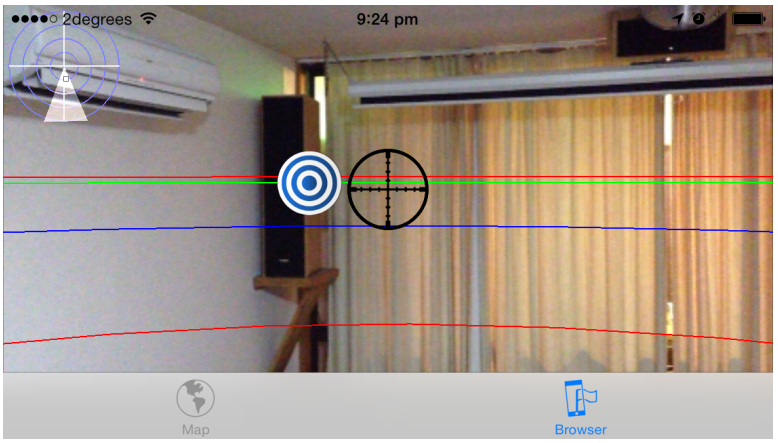

FIGURE 6.3: A screenshot from the user evaluation application.

The test was run on a battery powered iPhone 5 with a video resolution of 480×360. The results were saved to a CSV file which were processed using a Ruby script.

6.3.2 Results

9 users hit over 140 targets, and we have summarised our results based on the tracking method used. A full set of results can be found in Appendix D.

TABLE 6.3: User Study Performance Comparison

User ID	Tracking Method	Mean	S.D.	S.E.
All	Sensor Tracking	16.53s	7.02s	2.34s
All	Hybrid Tracking	7.09s	3.31s	1.10s

The results in Table 6.3 clearly show the benefit of our proposed hybrid tracking algorithm. A paired t-test is used to compare the different algorithms. There is a significant difference in the average task completion time for sensor tracking ($M = 16.53s$, $SD = 7.02s$) and hybrid tracking ($M = 7.09s$, $SD = 3.31s$) conditions: $t(8) = 5.0089$, $p < 0.0010$.

6.3.3 Feedback

Almost all feedback was positive and preferred the proposed tracking algorithm (blue targets). In particular, it was observed that users generally found the sensor tracking (red targets) frustrating due to misalignment and drift, making it hard to precisely hit the red targets. Our algorithm did not suffer from the same problems and was also significantly more precise and responsive to small changes in rotation.

"The red ones are really hard."

"If I had to pick one, I'd say blue was easier. Initially, both were quite good."

"It's quite hard, the red one. Blue was much easier."

"The blue one was easier, and it was more precise."

"It changed - at first the red ones were good, but at some point the red ones seemed to have their own personality, they would run away."

6.4 Summary

We have tested our proposed algorithm to evaluate it's accuracy, efficiency and usability on a mobile device. We found that it's performance was good on current generation hardware, and significantly faster than existing computer vision algorithms. Precision was also improved over pure sensor tracking in real world tests, and these results were confirmed by a user study.

Chapter 7

Conclusion

We have presented an efficient method to reduce jitter, enhance accuracy, and improve the outdoor AR experience on consumer-level mobile phones and tablets. Our hybrid algorithm combines sensor data and video frames to ensure reliable tracking, and is tolerant to both inertial and visual noise. The proposed scanline based vertical edge detector can be tuned for real-time performance on a variety of different input resolutions and hardware levels, and the fast integral sequence alignment algorithm can correct errors in the inertial sensor measurements efficiently. For its intended usage on mobile devices, our proposed hybrid approach is both faster than existing algorithms, such that it can run in real time with minimal overhead, and sufficiently accurate even in cases where existing purely vision-based algorithm could fail.

The sensor accuracy in consumer level hardware has improved considerably over the past several years and will continue to improve. During the course of this research, the amount of processing power has doubled three times, from the iPhone 4, to the iPhone 4S, iPhone 5 and now iPhone 5S with the motion co-processor. Our image alignment algorithm exploits this trend by being significantly more efficient when supplied the correct motion estimate. Ideally, as sensors continue to improve in accuracy, the implementation should incur proportionally less overhead.

Our results show that our algorithm is efficient, even at higher frame resolutions. Our effort to develop specific algorithms for mobile outdoor AR ultimately allowed us to improve the accuracy up to 25× in some cases with only 2-4ms in processing time per video frame on modern mobile devices. In comparison to existing computer vision algorithms, this is significantly faster.

Quantifiable comparisons of mobile outdoor AR algorithms are difficult due to the lack of open source implementations and testing methodologies. They often require the reimplementation of existing research, and published comparisons become less relevant as hardware progresses. To support future synthetic and real world evaluations in this area, we have made available, under a permissive open source licence, the tooling and data sets developed during this research. We hope that these tools will be embraced by the research community and allow for the accurate and up-to-date comparison of different tracking methods.

Practically speaking, the technology and knowledge developed over the past two years has fed into several commercial mobile projects, including DiscovAR, CityView-AR[28] for iOS, ColAR and FurniView[1]. However, despite the proprietary nature of these projects, we have published a significant amount of source code and designed systems to support future research in this area. We have confidence in the long term viability and value of this effort and will continue to maintain it for the foreseeable future. To ensure this, we have put in place an open source development methodology which encourages collaboration and shared responsibility. We look forward to seeing how it develops over the next few years, and hope to see it used in exciting new research and commercial projects.

7.1 Future Directions

Point cloud based algorithms and tracking can be used for precise positioning and registration tasks. Existing approaches for tracking camera pose and registering content have demonstrated the viability of this approach in small environments. We believe that point cloud tracking and registration can significantly improve the alignment of content in task-specific outdoor contexts, and thus it seems like the next logical step for improved outdoor AR. As such, we would like to support continued development of the Transform Flow toolkit and integrate existing libraries for point cloud tracking and analysis.

Android support is critical for practical deployment of Transform Flow based algorithms. We already have basic support for Android but further work is needed to integrate with the Transform Flow motion model interface. Most of the existing software is compatible with Android, but the actual browser code requires a significant investment of platform specific code. We would like to see this developed

in the future so that we maximise the value of the framework for a wide variety of mobile devices.

The current motion model only supports registration within a global frame of reference. We would like to expand this idea for local tracking and registration tasks. We would therefore like to implement frame-of-reference specific motion models which could then be run in parallel and used to combine different tracking and registration techniques into a single application.

Because Transform Flow is an open source project, other researchers have already expressed interest in continuing the work and developing additional functionality, including most of the above. We would like to continue keeping Transform Flow relevant and hope that the project continues to grow.

Appendix A

Fast Integer Sequence Alignment Implementation

```
struct Cost
{
  int offset;
  float error;

  std::size_t count;

  Cost(int _offset, float _error = 0) : offset(_offset), error(
     _error), count(0) {}

  bool operator>(const Cost & other) const
  { return error > other.error; }

  bool operator<(const Cost & other) const
  { return error < other.error; }

  bool operator==(const Cost & other) const
  { return this->offset == other.offset && this->error == other.
     error; }

  bool operator!=(const Cost & other) const
  { return !((*this) == other); }
```

```
  void add_error(float amount)
  { error += amount; }
};

static float error_bias(std::size_t difference)
{ return (difference * difference) / 2.0; }

template <typename SequenceT>
static Cost calculate_alignment_cost(const SequenceT & u, const
   SequenceT & v, int offset, int estimate, float best_error)
{
  Cost cost = {offset};

  std::size_t i = 0, j = 0;

  if (offset < 0)
    j = -offset;
  else
    i = offset;

  cost.error += error_bias(offset - estimate);

  while (i < u.size() && j < v.size())
  {
    auto d = float(u[i]) - float(v[j]);

    cost.add_error(d*d);

    i += 1, j += 1;
    cost.count += 1;

    if (cost.error > best_error) break;
  }

  return cost;
}

template <typename SequenceT>
```

```
Cost align_small(const SequenceT & u, const SequenceT & v, int
    estimate)
{
  Cost minimum_cost = calculate_alignment_cost(u, v, estimate,
      estimate, std::numeric_limits<float>::max());

  // Used to control expansion of the search space:
  int left = estimate - 1;
  int right = estimate + 1;

  // The bounds of the offset search:
  int right_bound = (int)u.size() / 2;
  int left_bound = -right_bound;

  while (left > left_bound || right < right_bound) {
    if (left > left_bound) {
      Cost cost = calculate_alignment_cost(u, v, left, estimate,
          minimum_cost.error);

      if (cost.error <= minimum_cost.error)
        minimum_cost = cost;

      left -= 1;
    }

    if (right < right_bound) {
      Cost cost = calculate_alignment_cost(u, v, right, estimate
          , minimum_cost.error);

      if (cost.error <= minimum_cost.error)
        minimum_cost = cost;

      right += 1;
    }
  }

  return minimum_cost;
}
```

Appendix B

Motion Model Implementation

This is a basic outline of the motion model code, including the currently specified sensor updates.

```
struct SensorUpdate {
  TimeT time_offset;

  // Used to keep track of debugging information relating to
     this sensor update.
  mutable std::vector<std::string> notes;
};

struct LocationUpdate : public SensorUpdate {
  double latitude, longitude, altitude;
  double horizontal_accuracy, vertical_accuracy;
};

struct HeadingUpdate : public SensorUpdate {
  // The axis which if that axis was pointing north, the true
     bearing would be 0. Defaults to <0, 1, 0>.
  Vec3 device_north;

  double magnetic_bearing, true_bearing;
};

struct MotionUpdate : public SensorUpdate {
  // In radians/second.
```

```
  Vec3 rotation_rate;
  Vec3 acceleration;
  Vec3 gravity;
};

struct ImageUpdate : public SensorUpdate {
  Ref<Image> image_buffer;

  // The horizontal field of view of the camera image updates:
  Radians<> field_of_view;
};

class MotionModel
{
  public:
    // Inputs:
    virtual void update(const LocationUpdate &) = 0;
    virtual void update(const HeadingUpdate &) = 0;
    virtual void update(const MotionUpdate &) = 0;
    virtual void update(const ImageUpdate &) = 0;

    // Outputs:
    virtual bool localization_valid() const;
    const Vec3 & gravity() const;
    const Vec3 & position() const;
    Radians<> bearing() const;
};
```

Appendix C

Video Stream Format

The documentation for the video stream format is available online[98]. Due to the dynamic nature of the transform flow platform, it is likely that it will grow over time to support additional types of hardware and sensors. However, the basic outline provided here should remain relevant and serve as a useful guide.

The format itself consists of a directory of images and a CSV log file. The general CSV format is as follows:

```
[sequence-number],[event-name],[event-arguments,]
```

There are several pre-defined events:

```
const char * GYROSCOPE = "Gyroscope";
const char * ACCELEROMETER = "Accelerometer";
const char * GRAVITY = "Gravity";
const char * MOTION = "Motion";
const char * LOCATION = "Location";
const char * HEADING = "Heading";
const char * FRAME = "Frame";
```

C.1 Motion Events

The motion event is a group of device sensor information at a single timestamp. It currently includes the `Gyroscope`, `Accelerometer` and `Gravity` events with the same timestamp and typically takes the form:

```
1,Gyroscope,[timestamp],[rotation.x],[rotation.y],[rotation.z]
2,Accelerometer,[timestamp],[acceleration.x],[acceleration.y],[
   acceleration.z]
3,Gravity,[timestamp],[gravity.x],[gravity.y],[gravity.z]
4,Motion,[timestamp]
```

The timestamp SHOULD be the same value for grouped motion events.

C.2 Location Events

The location event typically represents an update from the GPS and includes the position and accuracy of the update:

```
1,Location,[timestamp],[latitude],[longitude],[altitude],[
   horizontal_accuracy],[vertical_accuracy]
```

C.3 Heading Events

The heading event typically represents an update from the compass. It includes both the magnetic north and true north.

```
1,Heading,[timestamp],[magnetic_bearing],[true_bearing]
```

C.4 Frame Events

The frame event represents a camera frame captured and includes data as an external PNG file in the same directory as the log file.

```
1,Frame,[timestamp],[index]
```

The file in this case would be called `[index].png` or `[index].jpg`.

Appendix D

User Study Results

Each user was asked to hit between 10-20 targets, and the results are listed below.

Table D.1: Full User Study Results

User ID	Tracking Method	Mean	S.D.	S.E.
1	Sensor Tracking	16.5s	15.4s	4.7s
1	Hybrid Tracking	8.5s	4.0s	1.2s
2	Sensor Tracking	11.8s	11.0s	3.3s
2	Hybrid Tracking	7.2s	6.2s	2.0s
3	Sensor Tracking	33.2s	38.0s	12.0s
3	Hybrid Tracking	12.1s	20.4s	6.4s
4	Sensor Tracking	22.7s	18.2s	8.1s
4	Hybrid Tracking	5.7s	0.6s	0.3s
5	Sensor Tracking	22.2s	18.7s	7.1s
5	Hybrid Tracking	9.2s	6.5s	2.7s
6	Sensor Tracking	12.0s	12.2s	4.6s
6	Hybrid Tracking	9.8s	12.8s	5.2s
7	Sensor Tracking	16.9s	16.9s	6.4s
7	Hybrid Tracking	3.8s	0.8s	0.3s
8	Sensor Tracking	17.4s	15.6s	4.9s
8	Hybrid Tracking	4.0s	1.5s	0.5s
9	Hybrid Tracking	3.5s	2.1s	0.6s
9	Sensor Tracking	10.8s	7.9s	2.3s

Appendix E

Work Log

E.1 Late 2011: Ground Plane Detection

One of the first ideas we explored involved tracking feature points on the ground plane. Due to the speed and accuracy of the accelerometer and gyroscope, calculating the gravity vector is fast and easy to do. Using this information, a bird's eye visual representation of the ground plane can be extracted. Using this transform, feature points can be tracked on the ground plane and mapped to changes in bearing and position using existing algorithms such as optical flow[34].

We implemented a basic ground plane extraction algorithm, but after investigating this method, we concluded that this approach isn't viable in general. In many cases the accuracy of the camera wouldn't be good enough to track feature points (e.g. if the camera is pointing more than 45° up, the resolution of the ground plane is reduced significantly) and non-planar features caused significant artefacts in the final image.

This original implementation was running in real-time and was hard to test. Therefore, we decided to focus on tools data acquisition and analysis.

E.2 Late 2011: Data Acquisition Tools

In order to support the development and evaluation of new algorithms we decided to create data capture and analysis tools. These tools allow existing and new approaches to be tested in a controlled environment thus ensuring reproducibility of results. Specifically, we decided these tools would support our quantitative evaluation including comparisons between different algorithms.

The first tool we designed was a mobile data acquisition system (see Figure E.1) which captures directly from an iPhone's sensors, including the gyroscope, accelerometer, gravity and video frames. We used the existing ARBrowser application as a starting point (developed late 2010) and updated it to support real-time data logging and graphing.

FIGURE E.1: The data acquisition application running on an iPhone 4.

We captured several sample sequences and have used the data capture tool throughout our research to capture additional sequences for development, bug isolation and evaluation.

E.3 Early 2012: Data Analysis

After developing the data capture tool, we started thinking about how to analyse the data and design an algorithm. We started working on a desktop application for processing the captured data-sets and visualising them in 3D, such as in 5.3.

We used this tool to develop intuition and explore ideas. We could repeatedly replay the same data-set with different algorithms and check the results. This

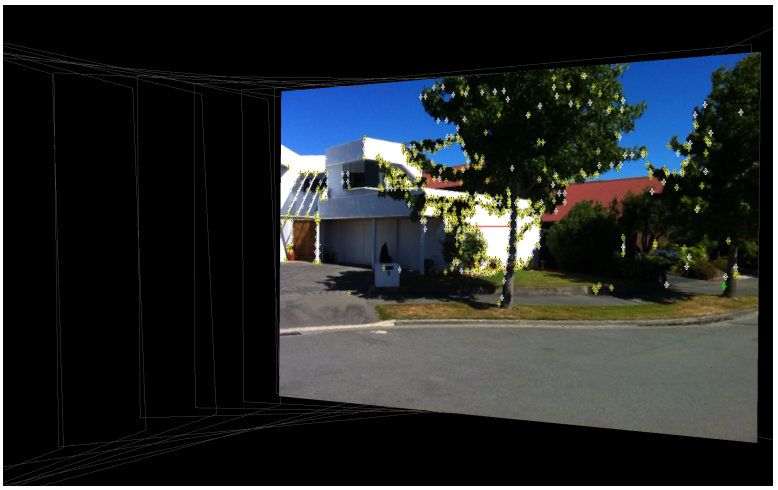

FIGURE E.2: The data visualisation application running on a laptop computer.

helped guide our designs for the proposed algorithm and through visual inspection of data processing and image analysis we have developed a good understanding of the strengths and weaknesses of different approaches.

E.4 Mid 2012: Transform Flow Implementation

Our first approach extracted key-points by using scanlines to detect changes in adjacent pixels, in some ways similar to a Harris Corner Detector[99], but it improves on typical feature registration by using the gravity vector for consistency across frames.

Using sensor data, a motion estimate between sequential frames could be calculated. Using camera intrinsics along with motion estimates allows us to formulate a per-pixel correspondence between image frames, however due to sensor inaccuracies, visual features are not always aligned from one frame to the next. We sought to reduce this error by aligning visual features between frames using the motion estimate as a predictive guide as to where features may have shifted.

To reduce the burden of extracting feature points every frame, we attempted to track edges across multiple frames. Some data-sets worked reasonably well, but others failed. Individual feature points could provide a good local estimate, but calculating a global change in position was still difficult.

Our preliminary results confirmed that traditional feature point tracking algorithms would not be able to run in real-time without significant modifications - building

pyramids, extracting feature points, and tracking them across frames were all incredibly expensive operations and existing software was not designed with mobile hardware in mind. Because of this, we decided that a good quality for a scalable algorithm would be the ability to adjust the amount of image processing required and get a proportional change in accuracy.

E.5 Mid 2012: ST Project

I travelled to Singapore for three weeks to work on an augmented reality navigation application (see E.3 for an example screenshot). Due to project requirements we were targeting iPhone hardware. We combined existing research from navigation and outdoor augmented reality to develop a dynamic navigation system that responded to the user's position and orientation.

FIGURE E.3: A screenshot from the Urban Navigation project developed as part of the ST Project.

This implementation provided practical insight into the limitations of existing sensor based tracking algorithms. We adjusted the design of the application so that error

in the global position would have less of an effect on the visualisation. In addition, a significant amount of effort was required to ensure that the navigation waypoint tracking was tolerant to both positional and orientation errors.

E.6 Late 2012: Teapot

After becoming significantly frustrated with the growing amount of source code and the challenges associated with building on multiple platforms, we started working on Teapot. The initial structure for Teapot had been developed as part of the Dream framework for OpenGL graphics, but the original implementation was not sufficient for use in other projects. By extracting out and refining the existing package management scripts from Dream, Teapot was born.

E.7 Late 2012: ColAR

As part of a commercial company grown out of HIT Lab NZ research, we have been implemented a typical natural feature tracking algorithm for planar based registration. While the goals of this work are significantly different from typical outdoor augmented reality, the algorithms and their use overlap in some areas and it was helpful to look at existing natural feature tracking algorithms and their efficient implementation.

The prototype for ColAR was designed to overlay 3D content on 2D planar colouring in pages. The initial registration step analyses feature point correspondence between a known 2D planar image and the input from the camera (see Figure E.4). Not all features match correctly, so RANSAC is used to refine the set of features by assuming planar correspondence.

Corresponding feature points are used to extract 3D translation and rotation, which allows virtual content to be aligned with planar surfaces in the real world.

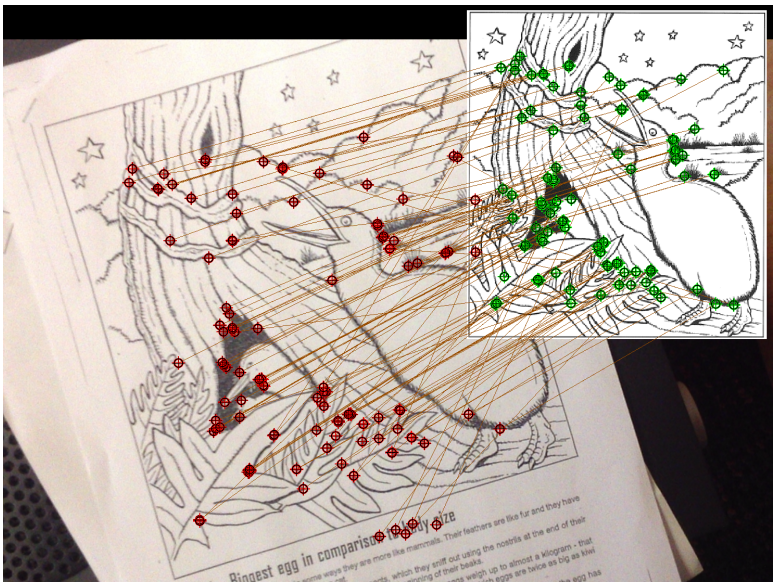

FIGURE E.4: Feature point correspondence using ORB binary feature points and FLANN matching.

E.8 Mid 2013: Refinement

We improved the visualisation platform and used it to debug and evaluate our proposed algorithm. Our previous implementation of the image alignment code was brute force. We formalised the FISA algorithm and developed several initial implementations to test the viability of the integral alignment approach.

E.9 Late 2013: Evaluation

We modified our existing browser to support the Transform Flow library's algorithms. We used this to develop and run a user study. Practical testing also revealed several areas where the algorithm could be improved, and testing on a variety of different camera resolutions was also useful.

We captured 8 additional data sets and used the visualisation tool to define trackpoints and evaluate our proposed algorithm.

Bibliography

[1] Adam Hutchinson, Samuel Williams, and Carl Watson. FurniView, 2013. URL http://furniview.co.nz.

[2] Hirokazu Kato. ARToolKit: library for vision-based augmented reality. *IEICE, PRMU*, pages 79–86, 2002.

[3] Samuel Williams. Dream framework, 2005. URL http://www.codeotaku.com/projects/dream.

[4] A Canclini, M Cesana, A Redondi, M Tagliasacchi, J Ascenso, and R Cilla. Evaluation of low-complexity visual feature detectors and descriptors.

[5] Ronald Azuma, Bruce Hoff, Howard Neely III, and Ron Sarfaty. A motion-stabilized outdoor augmented reality system. In *Virtual Reality, 1999. Proceedings., IEEE*, pages 252–259. IEEE, 1999.

[6] Mathias Mohring, Christian Lessig, and Oliver Bimber. Video see-through AR on consumer cell-phones. In *Proceedings of the 3rd IEEE/ACM International Symposium on Mixed and Augmented Reality*, ISMAR '04, pages 252–253, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2191-6. doi: 10.1109/ISMAR.2004.63.

[7] Daniel Wagner, Gerhard Reitmayr, Alessandro Mulloni, Tom Drummond, and Dieter Schmalstieg. Pose tracking from natural features on mobile phones. In *Proceedings of the 7th IEEE/ACM International Symposium on Mixed and Augmented Reality*, ISMAR '08, pages 125–134, Washington, DC, USA, 2008. IEEE Computer Society. ISBN 978-1-4244-2840-3. doi: 10.1109/ISMAR.2008.4637338.

[8] Bolan Jiang, Ulrich Neumann, and Suya You. A robust hybrid tracking system for outdoor augmented reality. In *Virtual Reality, 2004. Proceedings. IEEE*, pages 3–11. IEEE, 2004.

[9] WT Fong, Soh-Khim Ong, and Andrew YC Nee. Computer vision centric hybrid tracking for augmented reality in outdoor urban environments. In *Proceedings of the 8th International Conference on Virtual Reality Continuum and its Applications in Industry*, pages 185–190. ACM, 2009.

[10] Layar: Augmented reality, interactive print, 2013. URL http://www.layar.com.

[11] Wikitude: The world's leading augmented reality SDK, 2013. URL http://www.wikitude.com.

[12] Gerhard Schall, Daniel Wagner, Gerhard Reitmayr, Elise Taichmann, Manfred Wieser, Dieter Schmalstieg, and Bernhard Hofmann-Wellenhof. Global pose estimation using multi-sensor fusion for outdoor augmented reality. In *Proceedings of the 2009 8th IEEE International Symposium on Mixed and Augmented Reality*, ISMAR '09, pages 153–162, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-1-4244-5390-0. doi: 10.1109/ISMAR.2009.5336489.

[13] Ronald T Azuma. The challenge of making augmented reality work outdoors. *Mixed reality: Merging real and virtual worlds*, 1:379–390, 1999.

[14] Pierre Fite-Georgel. Is there a reality in industrial augmented reality? In *Mixed and Augmented Reality (ISMAR), 2011 10th IEEE International Symposium on*, pages 201–210. IEEE, 2011.

[15] Thomas Olsson, Tuula Kärkkäinen, Else Lagerstam, and Leena Ventä-Olkkonen. User evaluation of mobile augmented reality scenarios. *Journal of Ambient Intelligence and Smart Environments*, 4(1):29–47, 2012.

[16] James Wen, William Helton, and Mark Billinghurst. A study of user perception, interface performance, and actual usage of mobile pedestrian navigation aides. In *The 57th Annual Meeting of the Human Factors and Ergonomics Society*, 2013.

[17] Daniel Wagner, Alessandro Mulloni, Tobias Langlotz, and Dieter Schmalstieg. Real-time panoramic mapping and tracking on mobile phones. In *Proceedings of the 2010 IEEE Virtual Reality Conference*, VR '10, pages 211–218, Washington, DC, USA, 2010. IEEE Computer Society. ISBN 978-1-4244-6237-7. doi: 10.1109/VR.2010.5444786.

[18] Ivan E. Sutherland. A head-mounted three dimensional display. In *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*, AFIPS '68 (Fall, part I), pages 757–764, New York, NY, USA, 1968. ACM. doi: 10.1145/1476589.1476686.

[19] Steve Mann. An historical account of the 'wearcomp' and 'wearcam' inventions developed for applications in 'personal imaging'. In *Wearable Computers, 1997. Digest of Papers., First International Symposium on*, pages 66–73. IEEE, 1997.

[20] Louis B Rosenberg. The use of virtual fixtures as perceptual overlays to enhance operator performance in remote environments. Technical report, DTIC Document, 1992.

[21] Daniel Wagner and Dieter Schmalstieg. Making augmented reality practical on mobile phones, part 1. *IEEE Comput. Graph. Appl.*, 29(3):12–15, 2009. ISSN 0272-1716. doi: 10.1109/MCG.2009.46.

[22] Tobias Langlotz Daniel Wagner, Alessandro Mulloni and Dieter Schmalstieg. Real-time panoramic mapping and tracking on mobile phones. *IEEE Virtual Reality*, pages 211–218, 2010. ISSN 1087-8270.

[23] Gartner. Smartphone sales grew 46.5 percent in second quarter of 2013, 2013. URL http://www.gartner.com/newsroom/id/2573415.

[24] Georg Klein and David Murray. Parallel tracking and mapping on a camera phone. In *Proceedings of the 2009 8th IEEE International Symposium on Mixed and Augmented Reality*, ISMAR '09, pages 83–86, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-1-4244-5390-0. doi: 10.1109/ISMAR.2009.5336495.

[25] Daniel Wagner and Dieter Schmalstieg. Making augmented reality practical on mobile phones, part 2. *IEEE Comput. Graph. Appl.*, 29(4):6–9, 2009. ISSN 0272-1716. doi: 10.1109/MCG.2009.67.

[26] Paul A Zandbergen. Accuracy of iPhone locations: A comparison of assisted GPS, WiFi and cellular positioning. *Transactions in GIS*, 13(s1):5–25, 2009.

[27] Jeffrey R Blum, D Greencorn, and Jeremy R Cooperstock. Smartphone sensor reliability for augmented reality applications. In *Proc. MobiQuitous*, 2012.

[28] Gun A Lee, Andreas Dunser, Seungwon Kim, and Mark Billinghurst. CityViewAR: A mobile outdoor AR application for city visualization. In *Mixed and Augmented Reality (ISMAR-AMH), 2012 IEEE International Symposium on*, pages 57–64. IEEE, 2012.

[29] Michael G Wing, Aaron Eklund, and Loren D Kellogg. Consumer-grade global positioning system (GPS) accuracy and reliability. *Journal of Forestry*, 103(4): 169–173, 2005.

[30] T. Rossler, S. Rogge, and C. Hentschel. A case study: Mobile augmented reality system for visualization of large buildings. In *Consumer Electronics - Berlin (ICCE-Berlin), 2011 IEEE International Conference on*, pages 311–314, 2011. doi: 10.1109/ICCE-Berlin.2011.6031827.

[31] Ronald T Azuma et al. A survey of augmented reality. *Presence*, 6(4):355–385, 1997.

[32] Anil K Jain. *Fundamentals of digital image processing*, volume 3. Prentice-Hall Englewood Cliffs, 1989.

[33] Motilal Agrawal and Kurt Konolige. Real-time localization in outdoor environments using stereo vision and inexpensive GPS. In *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*, volume 3, pages 1063–1068. IEEE, 2006.

[34] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th international joint conference on Artificial intelligence - Volume 2*, IJCAI'81, pages 674–679, San Francisco, CA, USA, 1981. Morgan Kaufmann Publishers Inc.

[35] Carlo Tomasi and Takeo Kanade. Detection and tracking of point features. Technical report, International Journal of Computer Vision, 1991.

[36] Jianbo Shi and Carlo Tomasi. Good features to track. Technical report, Ithaca, NY, USA, 1993.

[37] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *Computer Vision–ECCV 2006*, pages 430–443. Springer, 2006.

[38] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *Proceedings of the 2011 International*

Conference on Computer Vision, ICCV '11, pages 2564–2571, Washington, DC, USA, 2011. IEEE Computer Society. ISBN 978-1-4577-1101-5. doi: 10.1109/ICCV.2011.6126544.

[39] Alexandre Alahi, Raphael Ortiz, and Pierre Vandergheynst. Freak: Fast retina keypoint. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 510–517. IEEE, 2012.

[40] Edward Rosten, Reid Porter, and Tom Drummond. Faster and better: A machine learning approach to corner detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(1):105–119, 2010.

[41] Adrian Clarke. *OPIRA: The Optical-flow Perspective Invariant Registration Augmentation and other improvements for Natural Feature Registration*. PhD thesis, University of Canterbury, 2009.

[42] Tom Botterill, Steven Mills, and Richard Green. Bag-of-words-driven single camera simultaneous localisation and mapping. *Journal of Field Robotics*, 2010.

[43] Samuel Picton Drake. *Converting GPS coordinates [phi, lambda, h] to navigation coordinates (ENU)*. DSTO Electronics and Surveillance Research Laboratory, 2002.

[44] Fabrizio Borgia, Maria De Marsico, Emanuele Panizzi, and Lorenzo Pietrangeli. ARMob - augmented reality for urban mobility in RMob. In *Proceedings of the International Working Conference on Advanced Visual Interfaces*, pages 258–261. ACM, 2012.

[45] Richard Stanaway. GDA94, ITRF, WGS84: What's the difference? working with dynamic datums. *Australia, April*, 2007.

[46] New Zealand Geodetic Datum 2000 (NZGD2000), 2013. URL http://www.linz.govt.nz/geodetic/datums-projections-heights/geodetic-datums/new-zealand-geodetic-datum-2000.

[47] Gerasimos G. Rigatos. Extended kalman and particle filtering for sensor fusion in motion control of mobile robots. *Math. Comput. Simul.*, 81:590–607, November 2010. ISSN 0378-4754. doi: 10.1016/j.matcom.2010.05.003.

[48] Andrew J Davison, Walterio W Mayol, and David W Murray. Real-time localization and mapping with wearable active vision. In *Mixed and Augmented Reality, 2003. Proceedings. The Second IEEE and ACM International Symposium on*, pages 18–27. IEEE, 2003.

[49] Gerhard Reitmayr and Tom W. Drummond. Going out: Robust tracking for outdoor augmented reality. In *Proc. ISMAR 2006*, pages 109–118, Santa Barbara, CA, USA, October 22–25 2006. IEEE and ACM, IEEE CS.

[50] Tao Guan, Liya Duan, Junqing Yu, Yongjian Chen, and Xu Zhang. Real-time camera pose estimation for wide-area augmented reality applications. *IEEE Comput. Graph. Appl.*, 31:56–68, May 2011. ISSN 0272-1716. doi: 10.1109/MCG.2010.23.

[51] Mustafa Ozuysal, Michael Calonder, Vincent Lepetit, and Pascal Fua. Fast keypoint recognition using random ferns. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32:448–461, March 2010. ISSN 0162-8828. doi: 10.1109/TPAMI.2009.23.

[52] O. Nestares, Y. Gat, H. Haussecker, and I. Kozintsev. Video stabilization to a global 3D frame of reference by fusing orientation sensor and image alignment data. In *Mixed and Augmented Reality (ISMAR), 2010 9th IEEE International Symposium on*, pages 257–258, 2010. doi: 10.1109/ISMAR.2010.5643595.

[53] Marci Meingast, Christopher Geyer, and Shankar Sastry. Geometric models of rolling-shutter cameras. *arXiv preprint cs/0503076*, 2005.

[54] Moshe Ben-Ezra and Shree K. Nayar. Motion-based motion deblurring. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26:689–698, June 2004. ISSN 0162-8828. doi: 10.1109/TPAMI.2004.1.

[55] Suya You, Ulrich Neumann, and Ronald Azuma. Hybrid inertial and vision tracking for augmented reality registration. In *Virtual Reality, 1999. Proceedings., IEEE*, pages 260–267. IEEE, 1999.

[56] Leif Oppermann and Jubilee Campus. An abstract location model for mobile games. In *GI Jahrestagung*, pages 1863–1872, 2009.

[57] Gerhard Reitmayr, Tobias Langlotz, Daniel Wagner, Alessandro Mulloni, Gerhard Schall, Dieter Schmalstieg, and Qi Pan. Simultaneous localization and mapping for augmented reality. In *Proceedings of the 2010 International*

Symposium on Ubiquitous Virtual Reality, ISUVR '10, pages 5–8, Washington, DC, USA, 2010. IEEE Computer Society. ISBN 978-0-7695-4124-2. doi: 10.1109/ISUVR.2010.12.

[58] Georg Klein and David Murray. Parallel tracking and mapping for small AR workspaces. In *Proceedings of the 2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, ISMAR '07, pages 1–10, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 978-1-4244-1749-0. doi: 10.1109/ISMAR.2007.4538852.

[59] Wei Tan, Haomin Liu, Zilong Dong, Guofeng Zhang, and Hujun Bao. Robust monocular slam in dynamic environments. In *Proceedings of the 2013 12th IEEE International Symposium on Mixed and Augmented Reality*, ISMAR '13, 2013.

[60] Gerhard Reitmayr and Tom Drummond. Initialisation for visual tracking in urban environments. In *Proc. ISMAR 2007*, pages 161–160, Nara, Japan, Nov. 13–16 2007.

[61] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point cloud library (pcl). In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1–4. IEEE, 2011.

[62] Clemens Arth, Alessandro Mulloni, and Dieter Schmalstieg. Exploiting sensors on mobile phones to improve wide-area localization. In *Pattern Recognition (ICPR), 2012 21st International Conference on*, pages 2152–2156. IEEE, 2012.

[63] Daniel Kurz and Selim Ben Himane. Inertial sensor-aligned visual feature descriptors. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 161–166. IEEE, 2011.

[64] Daniel Kurz and Selim Benhimane. Handheld augmented reality involving gravity measurements. *Computers & Graphics*, 36(7):866–883, 2012. ISSN 0097-8493. doi: 10.1016/j.cag.2012.03.038.

[65] Daniel Kurz and Selim Benhimane. Gravity-aware handheld augmented reality. In *Proceedings of the 2011 10th IEEE International Symposium on Mixed and Augmented Reality*, ISMAR '11, pages 111–120, Washington, DC, USA, 2011. IEEE Computer Society. ISBN 978-1-4577-2183-0. doi: 10.1109/ISMAR.2011.6092376.

[66] Suwon Lee, Yong-Ho Seo, and Hyun S Yang. Scalable building facade recognition and tracking for outdoor augmented reality. In *Information Technology Convergence*, pages 923–931. Springer, 2013.

[67] W. T. Fong, S. K. Ong, and A. Y. C. Nee. Computer vision centric hybrid tracking for augmented reality in outdoor urban environments. In *Proceedings of the 8th International Conference on Virtual Reality Continuum and its Applications in Industry*, VRCAI '09, pages 185–190, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-912-1. doi: 10.1145/1670252.1670292.

[68] Steffen Gauglitz, Tobias Höllerer, and Matthew Turk. Evaluation of interest point detectors and feature descriptors for visual tracking. *Int. J. Comput. Vision*, 94(3):335–360, September 2011. ISSN 0920-5691. doi: 10.1007/s11263-011-0431-5.

[69] Sebastian Lieberknecht, Selim Benhimane, Peter Meier, and Nassir Navab. A dataset and evaluation methodology for template-based tracking algorithms. In *Proceedings of the 2009 8th IEEE International Symposium on Mixed and Augmented Reality*, ISMAR '09, pages 145–151, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-1-4244-5390-0. doi: 10.1109/ISMAR.2009.5336487.

[70] Masayuki Hayashi, Itaru Kitahara, Yoshinari Kameda, and Y Ojta. A study of camera tracking evaluation on TrakMark data-set. In *Proceedings of "The 2nd International Workshop on AR/MR Registration, Tracking and Benchmarking" Workshop in conjunction with ISMAR11*, page 5, 2011.

[71] Dieter Schmalstieg, Anton Fuhrmann, Gerd Hesina, Zsolt Szalavári, L Miguel Encarnaçao, Michael Gervautz, and Werner Purgathofer. The studierstube augmented reality project. *Presence: Teleoperators and Virtual Environments*, 11(1):33–54, 2002.

[72] Gerhard Reitmayr and Dieter Schmalstieg. Opentracker: A flexible software design for three-dimensional interaction. *Virtual reality*, 9(1):79–92, 2005.

[73] LibreGeoSocial, 2013. URL http://www.libregeosocial.org/.

[74] DroidAR, 2013. URL https://github.com/bitstars/droidar.

[75] mixare, 2013. URL http://www.mixare.org.

[76] OpenCV - open computer vision library, 2013. URL http://opencv.org.

[77] Jack E Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems journal*, 4(1):25–30, 1965.

[78] Andres Huertas and Gerard Medioni. Detection of intensity changes with subpixel accuracy using laplacian-gaussian masks. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (5):651–664, 1986.

[79] Saul B Needleman and Christian D Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, 48(3):443–453, 1970.

[80] Udo Lieneweg. Automated optical extraction from line arrays of the alignment between microfabricated layers. *JPL Technical Report Server 1992+*, 1997.

[81] B Srinivasa Reddy and Biswanath N Chatterji. An FFT-based technique for translation, rotation, and scale-invariant image registration. *Image Processing, IEEE Transactions on*, 5(8):1266–1271, 1996.

[82] Richard J Fateman. When is FFT multiplication of arbitrary-precision polynomials practical? *University of California, Berkeley*, 2006.

[83] Li-Jen Mao and Sheau-Dong Lang. An empirical study of heap traversal and its applications. In *Proceedings of the 7th World Multi Conference on Systematic, Cybernatics and Informatics, July*, pages 27–30, 2003.

[84] Samuel Williams. Tranform Flow capture application for iOS, 2013. URL https://github.com/HITLabNZ/transform-flow-capture-ios.

[85] Alexander Pacha and Samuel Williams. Tranform Flow capture application for Android, 2013. URL https://github.com/HITLabNZ/transform-flow-capture-android.

[86] Samuel Williams. Tranform Flow visualisation application, 2013. URL https://github.com/HITLabNZ/transform-flow-visualisation.

[87] Masayuki Hayashi, Koji Makita, Takeshi Kurata, Itaru Kitahara, Yoshinari Kameda, and Yuichi Ohta. Projective indices for AR/MR benchmarking in TrakMark. 2013.

[88] Samuel Williams. Tranform Flow data sets, 2013. URL https://github.com/HITLabNZ/transform-flow-data.

[89] Samuel Williams. Tranform Flow browser application for iOS, 2013. URL https://github.com/HITLabNZ/transform-flow-browser-ios.

[90] Linus Torvalds and Junio Hamano. Git: Fast version control system, 2010. URL http://git-scm.com.

[91] William J Schroeder, Luis Ibáñez, and Kenneth M Martin. Software process: the key to developing robust, reusable and maintainable open-source software. In *Biomedical Imaging: Nano to Macro, 2004. IEEE International Symposium on*, pages 648–651. IEEE, 2004.

[92] CMake - cross platform make, 2013. URL http://www.cmake.org.

[93] CMake - how to create a ProjectConfig.cmake file, 2013. URL http://www.cmake.org/Wiki/CMake/Tutorials/How_to_create_a_ProjectConfig.cmake_file.

[94] Carliss Y Baldwin and K Clark. The option value of modularity in design. *Harvard NOM Research Paper*, pages 2–13, 2002.

[95] Teapot, 2013. URL http://www.kyusu.org.

[96] RubyGems, 2013. URL http://rubygems.org.

[97] Travis CI - free hosted continuous integration platform, 2013. URL https://travis-ci.org.

[98] Samuel Williams. Tranform Flow library, 2013. URL https://github.com/HITLabNZ/transform-flow.

[99] C. Harris and M. Stephens. A combined corner and edge detector. In *Proceedings of the 4th Alvey Vision Conference*, pages 147–151, 1988.